

Java Card TM 2.1 Application Programming Interface



Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300

Final Revision 1.1, June 7, 1999

Copyright © 1999 Sun Microsystems, Inc.

901 San Antonio Road, Palo Alto, CA 94303 USA

All rights reserved. Copyright in this document is owned by Sun Microsystems, Inc.

Sun Microsystems, Inc. (SUN) hereby grants to you at no charge a nonexclusive, nontransferable, worldwide, limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice the Java Card™ 2.1 API Specification ("Specification") to use the Specification for internal evaluation purposes only. Other than this limited license, you acquire no right, title, or interest in or to the Specification and you shall have no right to use the Specification for productive or commercial use.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-1(a).

SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, JavaSoft, JavaBeans, JDK, Java, Java Card, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, EmbeddedJava, PersonalJava, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultraserver, Where The Network Is Going, Sun WorkShop, XView, Java WorkShop, the Java Coffee Cup logo, and Visual Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

Java Card API

Table of Contents

Overview	1
Class Hierarchy	4
Package java.lang	6
Class ArithmeticException	9
Class ArrayIndexOutOfBoundsException	11
Class ArrayStoreException	13
Class ClassCastException	15
Class Exception	17
Class IndexOutOfBoundsException	19
Class NegativeArraySizeException	21
Class NullPointerException	23
Class Object	25
Class RuntimeException	27
Class SecurityException	29
Class Throwable	31
Package javacard.framework	33
Class AID	35
Class APDU	39
Class APDUException	51
Class Applet	56
Class CardException	63
Class CardRuntimeException	66
Interface ISO7816	69
Class ISOException	76
Class JCSysstem	78
Class OwnerPIN	87
Interface PIN	92
Class PINException	95
Interface Shareable	98
Class SystemException	99
Class TransactionException	103
Class UserException	107
Class Util	110
Package javacard.security	117
Class CryptoException	119
Interface DESKey	123
Interface DSAKey	125
Interface DSAPrivateKey	129
Interface DSAPublicKey	131
Interface Key	133

Class KeyBuilder	135
Class MessageDigest	141
Interface PrivateKey	146
Interface PublicKey	147
Interface RSAPrivateCrtKey	148
Interface RSAPrivateKey	155
Interface RSAPublicKey	158
Class RandomData	161
Interface SecretKey	164
Class Signature	165
Package javacardx.crypto	176
Class Cipher	177
Interface KeyEncryption	186
Index	188

Java Card™ 2.1 Platform API Specification

Final Revision 1.1

This document is the specification for the Java Card 2.1 Application Programming Interface.

See:

Description

Packages	
java.lang	Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.
javacard.framework	Provides framework of classes and interfaces for the core functionality of a Java Card applet.
javacard.security	Provides the classes and interfaces for the Java Card security framework.
javacardx.crypto	Extension package containing security classes and interfaces for export-controlled functionality.

This document is the specification for the Java Card 2.1 Application Programming Interface.

Java Card 2.1 API Notes

Referenced Standards

ISO - International Standards Organization

- Information Technology - Identification cards - integrated circuit cards with contacts: ISO 7816
- Information Technology - Security Techniques - Digital Signature Scheme Giving Message Recovery: ISO 9796
- Information Technology - Data integrity mechanism using a cryptographic check function employing a block cipher algorithm: ISO 9797
- Information technology - Security techniques - Digital signatures with appendix : ISO 14888

RSA Data Security, Inc.

- RSA Encryption Standard: PKCS #1 Version 2.0
- Password-Based Encryption Standard: PKCS #5 Version 1.5

EMV

- The EMV '96 ICC Specifications for Payments systems Version 3.0

IPSec

- The Internet Key Exchange (IKE) document RFC 2409 (STD 1)

Standard Names for Security and Crypto

- SHA (also SHA-1): Secure Hash Algorithm, as defined in Secure Hash Standard, NIST FIPS 180-1.
- MD5: The Message Digest algorithm RSA-MD5, as defined by RSA DSI in RFC 1321.
- RIPEMD-160 : as defined in ISO/IEC 10118-3:1998 Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions
- DSA: Digital Signature Algorithm, as defined in Digital Signature Standard, NIST FIPS 186.
- DES: The Data Encryption Standard, as defined by NIST in FIPS 46-1 and 46-2.
- RSA: The Rivest, Shamir and Adleman Asymmetric Cipher algorithm.

Parameter Checking

Policy

All Java Card API implementations must conform to the Java model of parameter checking. That is, the API code should not check for those parameter errors which the VM is expected to detect. These include all parameter errors, such as null pointers, index out of bounds, and so forth, that result in standard runtime exceptions. The runtime exceptions that are thrown by the Java Card VM are:

- `ArithmeticException`
- `ArrayStoreException`
- `ClassCastException`
- `IllegalArgumentException`
- `IllegalStateException`
- `IndexOutOfBoundsException`
- `ArrayIndexOutOfBoundsException`
- `NegativeArraySizeException`
- `NullPointerException`
- `SecurityException`

Exceptions to the Policy

In some cases, it may be necessary to explicitly check parameters. These exceptions to the policy are documented in the Java Card API specification. A Java Card API implementation must not perform parameter checking with the intent to avoid runtime exceptions, unless this is clearly specified by the Java Card API specification.

Note: If multiple erroneous input parameters exist, any one of several runtime exceptions will be thrown by the VM. Java programmers rely on this behavior, but they do not rely on getting a specific exception. It is not necessary (nor is it reasonable or practical) to document the precise error handling for all possible combinations of equivalence classes of erroneous inputs. The value of this behavior is that the logic error in the calling program is detected and exposed via the runtime exception mechanism, rather than being masked by a normal return.

Hierarchy For All Packages

Package Hierarchies:

java.lang, javacard.framework, javacard.security, javacardx.crypto

Class Hierarchy

- class java.lang.Object
 - class javacard.framework.AID
 - class javacard.framework.APDU
 - class javacard.framework.Applet
 - class javacardx.crypto.Cipher
 - class javacard.framework.JCSystem
 - class javacard.security.KeyBuilder
 - class javacard.security.MessageDigest
 - class javacard.framework.OwnerPIN (implements javacard.framework.PIN)
 - class javacard.security.RandomData
 - class javacard.security.Signature
 - class java.lang.Throwable
 - class java.lang.Exception
 - class javacard.framework.CardException
 - class javacard.framework.UserException
 - class java.lang.RuntimeException
 - class java.lang.ArithmeticException
 - class java.lang.ArrayStoreException
 - class javacard.framework.CardRuntimeException
 - class javacard.framework.APDUException
 - class javacard.security.CryptoException
 - class javacard.framework.ISOException
 - class javacard.framework.PINException
 - class javacard.framework.SystemException
 - class javacard.framework.TransactionException
 - class java.lang.ClassCastException
 - class java.lang.IndexOutOfBoundsException
 - class java.lang.ArrayIndexOutOfBoundsException
 - class java.lang.NegativeArraySizeException
 - class java.lang.NullPointerException
 - class java.lang.SecurityException
 - class javacard.framework.Util

Interface Hierarchy

- interface javacard.security.DSAKey
 - interface javacard.security.DSAPrivateKey(also extends javacard.security.PrivateKey)
 - interface javacard.security.DSAPublicKey(also extends javacard.security.PublicKey)
 - interface javacard.framework.ISO7816
 - interface javacard.security.Key
 - interface javacard.security.PrivateKey
 - interface javacard.security.DSAPrivateKey(also extends javacard.security.DSAKey)
 - interface javacard.security.RSAPrivateCrtKey
 - interface javacard.security.RSAPrivateKey
 - interface javacard.security.PublicKey
 - interface javacard.security.DSAPublicKey(also extends javacard.security.DSAKey)
 - interface javacard.security.RSAPublicKey
 - interface javacard.security.SecretKey
 - interface javacard.security.DESKey
 - interface javacardx.crypto.KeyEncryption
 - interface javacard.framework.PIN
 - interface javacard.framework.Shareable
-
-

Package java.lang

Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.

See:

Description

Class Summary	
Object	Class <code>Object</code> is the root of the Java Card class hierarchy.
Throwable	The <code>Throwable</code> class is the superclass of all errors and exceptions in the Java Card subset of the Java language.

Exception Summary	
ArithmeticException	A JCRE owned instance of <code>ArithmeticException</code> is thrown when an exceptional arithmetic condition has occurred.
ArrayIndexOutOfBoundsException	A JCRE owned instance of <code>IndexOutOfBoundsException</code> is thrown to indicate that an array has been accessed with an illegal index.
ArrayStoreException	A JCRE owned instance of <code>ArrayStoreException</code> is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.
ClassCastException	A JCRE owned instance of <code>ClassCastException</code> is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.
Exception	The class <code>Exception</code> and its subclasses are a form of <code>Throwable</code> that indicates conditions that a reasonable applet might want to catch.
IndexOutOfBoundsException	A JCRE owned instance of <code>IndexOutOfBoundsException</code> is thrown to indicate that an index of some sort (such as to an array) is out of range.
NegativeArraySizeException	A JCRE owned instance of <code>NegativeArraySizeException</code> is thrown if an applet tries to create an array with negative size.
NullPointerException	A JCRE owned instance of <code>NullPointerException</code> is thrown when an applet attempts to use null in a case where an object is required.
RuntimeException	<code>RuntimeException</code> is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine. A method is not required to declare in its throws clause any subclasses of <code>RuntimeException</code> that might be thrown during the execution of the method but not caught.
SecurityException	A JCRE owned instance of <code>SecurityException</code> is thrown by the Java Card Virtual Machine to indicate a security violation. This exception is thrown when an attempt is made to illegally access an object belonging to a another applet.

Package java.lang Description

Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.

java.lang

Class ArithmeticException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.ArithmeticException

```

```

public class ArithmeticException
extends RuntimeException

```

A JCRE owned instance of `ArithmeticException` is thrown when an exceptional arithmetic condition has occurred. For example, a "divide by zero" is an exceptional arithmetic condition.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

ArithmeticException() Constructs an <code>ArithmeticException</code> .	
--	--

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

ArithmeticException

public ArithmeticException()

Constructs an ArithmeticException.

java.lang**Class ArrayIndexOutOfBoundsException**

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.IndexOutOfBoundsException
                |
                +--java.lang.ArrayIndexOutOfBoundsException

```

```

public class ArrayIndexOutOfBoundsException
extends IndexOutOfBoundsException

```

A JCRE owned instance of `IndexOutOfBoundsException` is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

<code>ArrayIndexOutOfBoundsException()</code>	
--	--

Constructs an <code>ArrayIndexOutOfBoundsException</code> .	
---	--

Methods inherited from class java.lang.Object
--

<code>equals</code>

Constructor Detail

ArrayIndexOutOfBoundsException

public ArrayIndexOutOfBoundsException()

Constructs an ArrayIndexOutOfBoundsException.

java.lang**Class ArrayStoreException**

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.ArrayStoreException

```

```

public class ArrayStoreException
extends RuntimeException

```

A JCRE owned instance of `ArrayStoreException` is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an `ArrayStoreException`:

```

Object x[] = new AID[3];
x[0] = new OwnerPIN( (byte) 3, (byte) 8);

```

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

<code>ArrayStoreException()</code>	
Constructs an <code>ArrayStoreException</code> .	

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

ArrayStoreException

public ArrayStoreException()

Constructs an ArrayStoreException.

java.lang

Class ClassCastException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.ClassCastException

```

```

public class ClassCastException
extends RuntimeException

```

A JCRE owned instance of `ClassCastException` is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a `ClassCastException`:

```

Object x = new OwnerPIN( (byte)3, (byte)8);
JCSysytem.getAppletShareableInterfaceObject( (AID)x, (byte)5 );

```

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

ClassCastException() Constructs a <code>ClassCastException</code> .	
---	--

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

ClassCastException

public ClassCastException()

Constructs a ClassCastException.

java.lang

Class Exception

```

java.lang.Object
|
+---java.lang.Throwable
      |
      +---java.lang.Exception

```

Direct Known Subclasses:

CardException, RuntimeException

```

public class Exception
extends Throwable

```

The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable applet might want to catch.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

Exception() Constructs an <code>Exception</code> instance.	
--	--

Methods inherited from class java.lang.Object

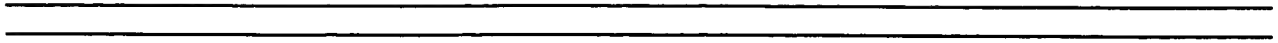
<code>equals</code>

Constructor Detail

Exception

```
public Exception()
```

Constructs an `Exception` instance.



java.lang**Class IndexOutOfBoundsException**

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.IndexOutOfBoundsException

```

Direct Known Subclasses:

ArrayIndexOutOfBoundsException

```

public class IndexOutOfBoundsException
extends RuntimeException

```

A JCRE owned instance of `IndexOutOfBoundsException` is thrown to indicate that an index of some sort (such as to an array) is out of range.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

IndexOutOfBoundsException() Constructs an <code>IndexOutOfBoundsException</code> .	
--	--

Methods inherited from class java.lang.Object
--

equals

Constructor Detail

IndexOutOfBoundsException

public IndexOutOfBoundsException()

Constructs an IndexOutOfBoundsException.

java.lang

Class NegativeArraySizeException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.NegativeArraySizeException

```

```

public class NegativeArraySizeException
extends RuntimeException

```

A JCRE owned instance of `NegativeArraySizeException` is thrown if an applet tries to create an array with negative size.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

NegativeArraySizeException()	
Constructs a <code>NegativeArraySizeException</code> .	

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

NegativeArraySizeException

`public NegativeArraySizeException()`

Constructs a NegativeArraySizeException.

java.lang

Class NullPointerException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.NullPointerException
  
```

public class NullPointerException
extends RuntimeException

A JCRE owned instance of `NullPointerException` is thrown when an applet attempts to use `null` in a case where an object is required. These include:

- Calling the instance method of a `null` object.
- Accessing or modifying the field of a `null` object.
- Taking the length of `null` as if it were an array.
- Accessing or modifying the slots of `null` as if it were an array.
- Throwing `null` as if it were a `Throwable` value.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

NullPointerException() Constructs a <code>NullPointerException</code> .	
---	--

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

NullPointerException

```
public NullPointerException()
```

Constructs a NullPointerException.

java.lang Class Object

java.lang.Object

public class Object

Class Object is the root of the Java Card class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

Object ()	
-----------	--

Method Summary

boolean	equals (Object obj) Compares two Objects for equality.
---------	--

Constructor Detail

Object

public Object()

Method Detail

equals

public boolean equals(Object obj)

Compares two Objects for equality.

The `equals` method implements an equivalence relation:

It is *reflexive*: for any reference value `x`, `x.equals(x)` should return `true`.

- It is *symmetric*: for any reference values `x` and `y`, `x.equals(y)` should return `true` if and only if `y.equals(x)` returns `true`.
- It is *transitive*: for any reference values `x`, `y`, and `z`, if `x.equals(y)` returns `true` and `y.equals(z)` returns `true`, then `x.equals(z)` should return `true`.
- It is *consistent*: for any reference values `x` and `y`, multiple invocations of `x.equals(y)` consistently return `true` or consistently return `false`.
- For any reference value `x`, `x.equals(null)` should return `false`.

The `equals` method for class `Object` implements the most discriminating possible equivalence relation on objects; that is, for any reference values `x` and `y`, this method returns `true` if and only if `x` and `y` refer to the same object (`x==y` has the value `true`).

Parameters:

`obj` - the reference object with which to compare.

Returns:

`true` if this object is the same as the `obj` argument; `false` otherwise.

java.lang

Class RuntimeException

```

java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException

```

Direct Known Subclasses:

ArithmeticException, ArrayStoreException, CardRuntimeException, ClassCastException, IndexOutOfBoundsException, NegativeArraySizeException, NullPointerException, SecurityException

```

public class RuntimeException
extends Exception

```

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine.

A method is not required to declare in its throws clause any subclasses of `RuntimeException` that might be thrown during the execution of the method but not caught.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

<code>RuntimeException()</code> Constructs a <code>RuntimeException</code> instance.	
--	--

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

RuntimeException

`public RuntimeException()`

Constructs a RuntimeException instance.

java.lang

Class SecurityException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--java.lang.SecurityException

```

public class SecurityException
extends RuntimeException

A JCRE owned instance of `SecurityException` is thrown by the Java Card Virtual Machine to indicate a security violation.

This exception is thrown when an attempt is made to illegally access an object belonging to a another applet. It may optionally be thrown by a Java Card VM implementation to indicate fundamental language restrictions, such as attempting to invoke a private method in another class.

For security reasons, the JCRE implementation may mute the card instead of throwing this exception.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

SecurityException() Constructs a <code>SecurityException</code> .	
---	--

Methods inherited from class java.lang.Object

<code>equals</code>

Constructor Detail

SecurityException

```
public SecurityException()
```

Constructs a SecurityException.

java.lang

Class Throwable

```
java.lang.Object
|
+---java.lang.Throwable
```

Direct Known Subclasses:
Exception

```
public class Throwable
extends Object
```

The Throwable class is the superclass of all errors and exceptions in the Java Card subset of the Java language. Only objects that are instances of this class (or of one of its subclasses) are thrown by the Java Card Virtual Machine or can be thrown by the Java throw statement. Similarly, only this class or one of its subclasses can be the argument type in a catch clause.

This Java Card class's functionality is a strict subset of the definition in the *Java Platform Core API Specification*.

Constructor Summary

Throwable() Constructs a new Throwable.	
---	--

Methods inherited from class java.lang.Object

equals

Constructor Detail

Throwable

```
public Throwable()

Constructs a new Throwable.
```



Package javacard.framework

Provides framework of classes and interfaces for the core functionality of a Java Card applet.

See:

Description

Interface Summary

<i>ISO7816</i>	ISO7816 encapsulates constants related to ISO 7816-3 and ISO 7816-4.
<i>PIN</i>	This interface represents a PIN.
<i>Shareable</i>	The Shareable interface serves to identify all shared objects.

Class Summary

AID	This class encapsulates the Application Identifier(AID) associated with an applet.
APDU	Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications.
Applet	This abstract class defines an applet in Java Card.
JCSys^{tem}	The JCSys ^{tem} class includes a collection of methods to control applet execution, resource management, atomic transaction management and inter-applet object sharing in Java Card.
OwnerPIN	This class represents an Owner PIN.
Util	The Ut ^{il} class contains common utility functions.

Exception Summary	
APDUException	APDUException represents an APDU related exception.
CardException	The CardException class defines a field <code>reason</code> and two accessor methods <code>getReason()</code> and <code>setReason()</code> .
CardRuntimeException	The CardRuntimeException class defines a field <code>reason</code> and two accessor methods <code>getReason()</code> and <code>setReason()</code> .
ISOException	ISOException class encapsulates an ISO 7816-4 response status word as its <code>reason</code> code.
PINException	PINException represents a OwnerPIN class access-related exception.
SystemException	SystemException represents a JCSysSystem class related exception.
TransactionException	TransactionException represents an exception in the transaction subsystem.
UserException	UserException represents a User exception.

Package javacard.framework Description

Provides framework of classes and interfaces for the core functionality of a Java Card applet.

javacard.framework

Class AID

```
java.lang.Object
|
+---javacard.framework.AID
```

public final class **AID**
 extends Object

This class encapsulates the Application Identifier(AID) associated with an applet. An AID is defined in ISO 7816-5 to be a sequence of bytes between 5 and 16 bytes in length.

The JCRE creates instances of AID class to identify and manage every applet on the card. Applets need not create instances of this class. An applet may request and use the JCRE owned instances to identify itself and other applet instances.

JCRE owned instances of AID are permanent JCRE Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

An applet instance can obtain a reference to JCRE owned instances of its own AID object by using the `JCSystem.getAID()` method and another applet's AID object via the `JCSystem.lookupAID()` method.

An applet uses AID instances to request to share another applet's object or to control access to its own shared object from another applet. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:

`JCSystem`, `SystemException`

Constructor Summary

AID(byte[] bArray, short offset, byte length)

The JCRE uses this constructor to create a new AID instance encapsulating the specified AID bytes.

Method Summary

boolean	equals (byte[] bArray, short offset, byte length) Checks if the specified AID bytes in bArray are the same as those encapsulated in this AID object.
boolean	equals (Object anObject) Compares the AID bytes in this AID instance to the AID bytes in the specified object.
byte	getBytes (byte[] dest, short offset) Called to get the AID bytes encapsulated within AID object.
boolean	partialEquals (byte[] bArray, short offset, byte length) Checks if the specified partial AID byte sequence matches the first length bytes of the encapsulated AID bytes within this AID object.
boolean	RIDEquals (AID otherAID) Checks if the RID (National Registered Application provider identifier) portion of the encapsulated AID bytes within the otherAID object matches that of this AID object.

Constructor Detail

AID

```
public AID(byte[] bArray,
           short offset,
           byte length)
    throws SystemException
```

The JCRE uses this constructor to create a new AID instance encapsulating the specified AID bytes.

Parameters:

- bArray - the byte array containing the AID bytes.
- offset - the start of AID bytes in bArray.
- length - the length of the AID bytes in bArray.

Throws:

SystemException - with the following reason code:

- `SystemException.ILLEGAL_VALUE` if the length parameter is less than 5 or greater than 16.

Method Detail

getBytes

```
public byte getBytes(byte[] dest,
                    short offset)
```

Called to get the AID bytes encapsulated within AID object.

Parameters:

dest - byte array to copy the AID bytes.
offset - within dest where the AID bytes begin.

Returns:

the length of the AID bytes.

equals

```
public boolean equals(Object anObject)
```

Compares the AID bytes in this AID instance to the AID bytes in the specified object. The result is true if and only if the argument is not null and is an AID object that encapsulates the same AID bytes as this object.

This method does not throw NullPointerException.

Parameters:

anObject - the object to compare this AID against.

Returns:

true if the AID byte values are equal, false otherwise.

Overrides:

equals in class Object

equals

```
public boolean equals(byte[] bArray,
                    short offset,
                    byte length)
```

Checks if the specified AID bytes in bArray are the same as those encapsulated in this AID object. The result is true if and only if the bArray argument is not null and the AID bytes encapsulated in this AID object are equal to the specified AID bytes in bArray.

This method does not throw NullPointerException.

Parameters:

bArray - containing the AID bytes
offset - within bArray to begin
length - of AID bytes in bArray

Returns:

true if equal, false otherwise.

partialEquals

```
public boolean partialEquals(byte[] bArray,  
                             short offset,  
                             byte length)
```

Checks if the specified partial AID byte sequence matches the first `length` bytes of the encapsulated AID bytes within this AID object. The result is `true` if and only if the `bArray` argument is not null and the input `length` is less than or equal to the length of the encapsulated AID bytes within this AID object and the specified bytes match.

This method does not throw `NullPointerException`.

Parameters:

`bArray` - containing the partial AID byte sequence

`offset` - within `bArray` to begin

`length` - of partial AID bytes in `bArray`

Returns:

`true` if equal, `false` otherwise.

RIDEquals

```
public boolean RIDEquals(AID otherAID)
```

Checks if the RID (National Registered Application provider identifier) portion of the encapsulated AID bytes within the `otherAID` object matches that of this AID object. The first 5 bytes of an AID byte sequence is the RID. See ISO 7816-5 for details. The result is `true` if and only if the argument is not null and is an AID object that encapsulates the same RID bytes as this object.

This method does not throw `NullPointerException`.

Parameters:

`otherAID` - the AID to compare against.

Returns:

`true` if the RID bytes match, `false` otherwise.

javacard.framework

Class APDU

```
java.lang.Object
|
+--javacard.framework.APDU
```

```
public final class APDU
extends Object
```

Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications. The format of the APDU is defined in ISO specification 7816-4.

This class only supports messages which conform to the structure of command and response defined in ISO 7816-4. The behavior of messages which use proprietary structure of messages (for example with header CLA byte in range 0xD0-0xFE) is undefined. This class does not support extended length fields.

The APDU object is owned by the JCRE. The APDU class maintains a byte array buffer which is used to transfer incoming APDU header and data bytes as well as outgoing data. The buffer length must be at least 37 bytes (5 bytes of header and 32 bytes of data). The JCRE must zero out the APDU buffer before each new message received from the CAD.

The JCRE designates the APDU object as a temporary JCRE Entry Point Object (See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details). A temporary JCRE Entry Point Object can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

The JCRE similarly marks the APDU buffer as a global array (See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details). A global array can be accessed from any applet context. References to global arrays cannot be stored in class variables or instance variables or array components.

The applet receives the APDU instance to process from the JCRE in the `Applet.process (APDU)` method, and the first five bytes [CLA, INS, P1, P2, P3] are available in the APDU buffer.

The APDU class API is designed to be transport protocol independent. In other words, applets can use the same APDU methods regardless of whether the underlying protocol in use is T=0 or T=1 (as defined in ISO 7816-3).

The incoming APDU data size may be bigger than the APDU buffer size and may therefore need to be read in portions by the applet. Similarly, the outgoing response APDU data size may be bigger than the APDU buffer size and may need to be written in portions by the applet. The APDU class has methods to facilitate this.

For sending large byte arrays as response data, the APDU class provides a special method `sendBytesLong ()` which manages the APDU buffer.

```

// The purpose of this example is to show most of the methods
// in use and not to depict any particular APDU processing

public void process(APDU apdu){
    // ...
    byte[] buffer = apdu.getBuffer();
    byte cla = buffer[ISO7816.OFFSET_CLA];
    byte ins = buffer[ISO7816.OFFSET_INS];
    ...
    // assume this command has incoming data
    // Lc tells us the incoming apdu command length
    short bytesLeft = (short) (buffer[ISO7816.OFFSET_LC] & 0x00FF);
    if (bytesLeft < (short)55) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );

    short readCount = apdu.setIncomingAndReceive();
    while ( bytesLeft > 0){
        // process bytes in buffer[5] to buffer[readCount+4];
        bytesLeft -= readCount;
        readCount = apdu.receiveBytes ( ISO7816.OFFSET_CDATA );
    }
    //
    //...
    //
    // Note that for a short response as in the case illustrated here
    // the three APDU method calls shown : setOutgoing(),setOutgoingLength() & sendBytes()
    // could be replaced by one APDU method call : setOutgoingAndSend().

    // construct the reply APDU
    short le = apdu.setOutgoing();
    if (le < (short)2) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH );
    apdu.setOutgoingLength( (short)3 );

    // build response data in apdu.buffer[ 0.. outCount-1 ];
    buffer[0] = (byte)1; buffer[1] = (byte)2; buffer[3] = (byte)3;
    apdu.sendBytes ( (short)0 , (short)3 );
    // return good complete status 90 00
}

```

See Also:

APDUException, ISOException

Field Summary

static byte	PROTOCOL_T0 ISO 7816 transport protocol type T=0
static byte	PROTOCOL_T1 ISO 7816 transport protocol type T=1

Method Summary

byte[]	getBuffer() Returns the APDU buffer byte array.
static short	getInBlockSize() Returns the configured incoming block size. In T=1 protocol, this corresponds to IFSC (information field size for ICC), the maximum size of incoming data blocks into the card. In T=0 protocol, this method returns 1.
byte	getNAD() In T=1 protocol, this method returns the Node Address byte, NAD. In T=0 protocol, this method returns 0.
static short	getOutBlockSize() Returns the configured outgoing block size. In T=1 protocol, this corresponds to IFSD (information field size for interface device), the maximum size of outgoing data blocks to the CAD. In T=0 protocol, this method returns 258 (accounts for 2 status bytes).
static byte	getProtocol() Returns the ISO 7816 transport protocol type, T=1 or T=0 in progress.
short	receiveBytes(short bOff) Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset bOff. Gets all the remaining bytes if they fit.
void	sendBytes(short bOff, short len) Sends len more bytes from APDU buffer at specified offset bOff.
void	sendBytesLong(byte[] outData, short bOff, short len) Sends len more bytes from outData byte array starting at specified offset bOff.
short	setIncomingAndReceive() This is the primary receive method.
short	setOutgoing() This method is used to set the data transfer direction to outbound and to obtain the expected length of response (Le).
void	setOutgoingAndSend(short bOff, short len) This is the "convenience" send method.
void	setOutgoingLength(short len) Sets the actual length of response data.
short	setOutgoingNoChaining() This method is used to set the data transfer direction to outbound without using BLOCK CHAINING(See ISO 7816-3/4) and to obtain the expected length of response (Le).
static void	waitExtension() Requests additional processing time from CAD.

Methods inherited from class java.lang.Object
--

<code>equals</code>

Field Detail

PROTOCOL_T0

`public static final byte PROTOCOL_T0`

ISO 7816 transport protocol type T=0

PROTOCOL_T1

`public static final byte PROTOCOL_T1`

ISO 7816 transport protocol type T=1

Method Detail

getBuffer

`public byte[] getBuffer()`

Returns the APDU buffer byte array.

Notes:

- *References to the APDU buffer byte array cannot be stored in class variables or instance variables or array components. See Java Card Runtime Environment (JCRE) 2.1 Specification for details.*

Returns:

byte array containing the APDU buffer

getInBlockSize

`public static short getInBlockSize()`

Returns the configured incoming block size. In T=1 protocol, this corresponds to IFSC (information field size for ICC), the maximum size of incoming data blocks into the card. In T=0 protocol, this method returns 1. IFSC is defined in ISO 7816-3.

This information may be used to ensure that there is enough space remaining in the APDU buffer when `receiveBytes()` is invoked.

Notes:

- On `receiveBytes()` the `bOff` param should account for this potential blocksize.

Returns:

incoming block size setting.

See Also:

`receiveBytes(short)`

getOutBlockSize

```
public static short getOutBlockSize()
```

Returns the configured outgoing block size. In T=1 protocol, this corresponds to IFSD (information field size for interface device), the maximum size of outgoing data blocks to the CAD. In T=0 protocol, this method returns 258 (accounts for 2 status bytes). IFSD is defined in ISO 7816-3.

This information may be used prior to invoking the `setOutgoingLength()` method, to limit the length of outgoing messages when BLOCK CHAINING is not allowed.

Notes:

- On `setOutgoingLength()` the `len` param should account for this potential blocksize.

Returns:

outgoing block size setting.

See Also:

`setOutgoingLength(short)`

getProtocol

```
public static byte getProtocol()
```

Returns the ISO 7816 transport protocol type, T=1 or T=0 in progress.

Returns:

the protocol type in progress. One of `PROTOCOL_T0`, `PROTOCOL_T1` listed above.

getNAD

```
public byte getNAD()
```

In T=1 protocol, this method returns the Node Address byte, NAD. In T=0 protocol, this method returns 0. This may be used as additional information to maintain multiple contexts.

Returns:

NAD transport byte as defined in ISO 7816-3.

setOutgoing

```
public short setOutgoing()  
            throws APDUException
```

This method is used to set the data transfer direction to outbound and to obtain the expected length of response (Le).

Notes.

- *Any remaining incoming data will be discarded.*
- *In T=0 (Case 4) protocol, this method will return 256.*

Returns:

Le, the expected length of response.

Throws:

APDUException - with the following reason codes:

- `APDUException.ILLEGAL_USE` if this method or `setOutgoingNoChaining()` method already invoked.
- `APDUException.IO_ERROR` on I/O error.

setOutgoingNoChaining

```
public short setOutgoingNoChaining()  
            throws APDUException
```

This method is used to set the data transfer direction to outbound without using BLOCK CHAINING(See ISO 7816-3/4) and to obtain the expected length of response (Le). This method should be used in place of the `setOutgoing()` method by applets which need to be compatible with legacy CAD/terminals which do not support ISO 7816-3/4 defined block chaining. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Notes.

- *Any remaining incoming data will be discarded.*
- *In T=0 (Case 4) protocol, this method will return 256.*
- *When this method is used, the `waitExtension()` method cannot be used.*
- *In T=1 protocol, retransmission on error may be restricted.*
- *In T=0 protocol, the outbound transfer must be performed without using response status chaining.*
- *In T=1 protocol, the outbound transfer must not set the More(M) Bit in the PCB of the I block. See ISO 7816-3.*

Returns:

Le, the expected length of response data.

Throws:

APDUException - with the following reason codes:

- `APDUException.ILLEGAL_USE` if this method or `setOutgoing()` method already invoked.

APDUException.IO_ERROR on I/O error.

setOutgoingLength

```
public void setOutgoingLength(short len)
           throws APDUException
```

Sets the actual length of response data. Default is 0.

Note:

- In T=0 (Case 2&4) protocol, the length is used by the JCRE to prompt the CAD for GET RESPONSE commands.

Parameters:

len - the length of response data.

Throws:

APDUException - with the following reason codes:

- APDUException.ILLEGAL_USE if setOutgoing() not called or this method already invoked.
- APDUException.BAD_LENGTH if len is greater than 256 or if non BLOCK CHAINED data transfer is requested and len is greater than (IFSD-2), where IFSD is the Outgoing Block Size. The -2 accounts for the status bytes in T=1.
- APDUException.IO_ERROR on I/O error.

See Also:

getOutBlockSize()

receiveBytes

```
public short receiveBytes(short bOff)
           throws APDUException
```

Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset bOff.

Gets all the remaining bytes if they fit.

Notes:

- The space in the buffer must allow for incoming block size.
- In T=1 protocol, if all the remaining bytes do not fit in the buffer, this method may return less bytes than the maximum incoming block size (IFSC).
- In T=0 protocol, if all the remaining bytes do not fit in the buffer, this method may return less than a full buffer of bytes to optimize and reduce protocol overhead.
- In T=1 protocol, if this method throws an APDUException with T1_IFD_ABORT reason code, the JCRE will restart APDU command processing using the newly received command. No more input data can be received. No output data can be transmitted. No error status response can be returned.

Parameters:

bOff - the offset into APDU buffer.

Returns:

number of bytes read. Returns 0 if no bytes are available.

Throws:

APDUException - with the following reason codes:

- APDUException.ILLEGAL_USE if setIncomingAndReceive() not called or if setOutgoing() or setOutgoingNoChaining() previously invoked.
- APDUException.BUFFER_BOUNDS if not enough buffer space for incoming block size.
- APDUException.IO_ERROR on I/O error.
- APDUException.T1_IFD_ABORT if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

See Also:

getInBlockSize()

setIncomingAndReceive

```
public short setIncomingAndReceive()  
            throws APDUException
```

This is the primary receive method. Calling this method indicates that this APDU has incoming data. This method gets as many bytes as will fit without buffer overflow in the APDU buffer following the header. It gets all the incoming bytes if they fit.

Notes:

- *In T=0 (Case 3&4) protocol, the P3 param is assumed to be Lc.*
- *Data is read into the buffer at offset 5.*
- *In T=1 protocol, if all the incoming bytes do not fit in the buffer, this method may return less bytes than the maximum incoming block size (IFSC).*
- *In T=0 protocol, if all the incoming bytes do not fit in the buffer, this method may return less than a full buffer of bytes to optimize and reduce protocol overhead.*
- *This method sets the transfer direction to be inbound and calls receiveBytes(5).*
- *This method may only be called once in a Applet.process() method.*

Returns:

number of bytes read. Returns 0 if no bytes are available.

Throws:

APDUException - with the following reason codes:

- APDUException.ILLEGAL_USE if setIncomingAndReceive() already invoked or if setOutgoing() or setOutgoingNoChaining() previously invoked.
 - APDUException.IO_ERROR on I/O error.
 - APDUException.T1_IFD_ABORT if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.
-

sendBytes

```
public void sendBytes(short bOff,
                     short len)
    throws APDUException
```

Sends `len` more bytes from APDU buffer at specified offset `bOff`.

If the last part of the response is being sent by the invocation of this method, the APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD. Requiring that the buffer not be altered allows the implementation to reduce protocol overhead by transmitting the last part of the response along with the status bytes.

Notes:

- If `setOutgoingNoChaining()` was invoked, output block chaining must not be used.
- In T=0 protocol, if `setOutgoingNoChaining()` was invoked, *Le* bytes must be transmitted before response status is returned.
- In T=0 protocol, if this method throws an `APDUException` with `NO_T0_GETRESPONSE` reason code, the JCRE will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.
- In T=1 protocol, if this method throws an `APDUException` with `T1_IFD_ABORT` reason code, the JCRE will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.

Parameters:

`bOff` - the offset into APDU buffer.

`len` - the length of the data in bytes to send.

Throws:

`APDUException` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLen()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` previously thrown.
- `APDUException.BUFFER_BOUNDS` if the sum of `bOff` and `len` exceeds the buffer size.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.NO_T0_GETRESPONSE` if T=0 protocol is in use and the CAD does not respond to response status with GET RESPONSE command.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

See Also:

`setOutgoing()`, `setOutgoingNoChaining()`

sendBytesLong

```
public void sendBytesLong(byte[] outData,  
                          short bOff,  
                          short len)  
    throws APDUException
```

Sends len more bytes from outData byte array starting at specified offset bOff.

If the last of the response is being sent by the invocation of this method, the APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD. Requiring that the buffer not be altered allows the implementation to reduce protocol overhead by transmitting the last part of the response along with the status bytes.

The JCIRE may use the APDU buffer to send data to the CAD.

Notes:

- If `setOutgoingNoChaining()` was invoked, output block chaining must not be used.
- In T=0 protocol, if `setOutgoingNoChaining()` was invoked, Le bytes must be transmitted before response status is returned.
- In T=0 protocol, if this method throws an `APDUException` with `NO_T0_GETRESPONSE` reason code, the JCIRE will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.
- In T=1 protocol, if this method throws an `APDUException` with `T1_IFD_ABORT` reason code, the JCIRE will restart APDU command processing using the newly received command. No more output data can be transmitted. No error status response can be returned.

Parameters:

outData - the source data byte array.
bOff - the offset into OutData array.
len - the bytelength of the data to send.

Throws:

`APDUException` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoingLen()` not called or `setOutgoingAndSend()` previously invoked or response byte count exceeded or if `APDUException.NO_T0_GETRESPONSE` previously thrown.
- `APDUException.IO_ERROR` on I/O error.
- `APDUException.NO_T0_GETRESPONSE` if T=0 protocol is in use and CAD does not respond to response status with GET RESPONSE command.
- `APDUException.T1_IFD_ABORT` if T=1 protocol is in use and the CAD sends in an ABORT S-Block command to abort the data transfer.

See Also:

`setOutgoing()`, `setOutgoingNoChaining()`

setOutgoingAndSend

```
public void setOutgoingAndSend(short bOff,
                               short len)
    throws APDUException
```

This is the "convenience" send method. It provides for the most efficient way to send a short response which fits in the buffer and needs the least protocol overhead. This method is a combination of `setOutgoing()`, `setOutgoingLength(len)` followed by `sendBytes (bOff, len)`. In addition, once this method is invoked, `sendBytes()` and `sendBytesLong()` methods cannot be invoked and the APDU buffer must not be altered.

Sends `len` byte response from the APDU buffer at starting specified offset `bOff`.

Notes:

- *No other APDU send methods can be invoked.*
- *The APDU buffer must not be altered. If the data is altered, incorrect output may be sent to the CAD.*
- *The actual data transmission may only take place on return from `Applet.process()`*

Parameters:

`bOff` - the offset into APDU buffer.

`len` - the bytelength of the data to send.

Throws:

`APDUException` - with the following reason codes:

- `APDUException.ILLEGAL_USE` if `setOutgoing()` or `setOutgoingAndSend()` previously invoked or response byte count exceeded.
- `APDUException.IO_ERROR` on I/O error.

waitExtension

```
public static void waitExtension()
    throws APDUException
```

Requests additional processing time from CAD. The implementation should ensure that this method needs to be invoked only under unusual conditions requiring excessive processing times.

Notes:

- *In T=0 protocol, a NULL procedure byte is sent to reset the work waiting time (see ISO 7816-3).*
- *In T=1 protocol, the implementation needs to request the same T=0 protocol work waiting time quantum by sending a T=1 protocol request for wait time extension(see ISO 7816-3).*
- *If the implementation uses an automatic timer mechanism instead, this method may do nothing.*

Throws:

`APDUException` - with the following reason codes:

`APDUException.ILLEGAL_USE` if `setOutgoingNoChaining()` previously

invoked.

APDUException.IO_ERROR on I/O error.

javacard.framework

Class APDUException

```
java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException
|
+--javacard.framework.CardRuntimeException
|
+--javacard.framework.APDUException
```

public class APDUException
extends CardRuntimeException

APDUException represents an APDU related exception.

The APDU class throws JCRE owned instances of APDUException.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:
APDU

Field Summary

static short	BAD_LENGTH This reason code is used by the <code>APDU.setOutgoingLength()</code> method to indicate that the length parameter is greater than 256 or if non BLOCK CHAINED data transfer is requested and <code>len</code> is greater than (IFSD-2), where IFSD is the Outgoing Block Size.
static short	BUFFER_BOUNDS This reason code is used by the <code>APDU.sendBytes()</code> method to indicate that the sum of buffer offset parameter and the byte length parameter exceeds the APDU buffer size.
static short	ILLEGAL_USE This <code>APDUException</code> reason code indicates that the method should not be invoked based on the current state of the APDU.
static short	IO_ERROR This reason code indicates that an unrecoverable error occurred in the I/O transmission layer.
static short	NO_T0_GETRESPONSE This reason code indicates that during T=0 protocol, the CAD did not return a GET RESPONSE command in response to a <61xx> response status to send additional data.
static short	T1_IFD_ABORT This reason code indicates that during T=1 protocol, the CAD returned an ABORT S-Block command and aborted the data transfer.

Constructor Summary

APDUException (short reason) Constructs an <code>APDUException</code> .	
---	--

Method Summary

static void	throwIt (short reason) Throws the JCRC owned instance of <code>APDUException</code> with the specified reason.
-------------	--

Methods inherited from class <code>javacard.framework.CardRuntimeException</code>
--

<code>getReason</code> , <code>setReason</code>

Methods inherited from class <code>java.lang.Object</code>

<code>equals</code>

Field Detail

ILLEGAL_USE

```
public static final short ILLEGAL_USE
```

This `APDUException` reason code indicates that the method should not be invoked based on the current state of the APDU.

BUFFER_BOUNDS

```
public static final short BUFFER_BOUNDS
```

This reason code is used by the `APDU.sendBytes()` method to indicate that the sum of buffer offset parameter and the byte length parameter exceeds the APDU buffer size.

BAD_LENGTH

```
public static final short BAD_LENGTH
```

This reason code is used by the `APDU.setOutgoingLength()` method to indicate that the length parameter is greater than 256 or if non BLOCK CHAINED data transfer is requested and `len` is greater than (IFSD-2), where IFSD is the Outgoing Block Size.

IO_ERROR

```
public static final short IO_ERROR
```

This reason code indicates that an unrecoverable error occurred in the I/O transmission layer.

NO_T0_GETRESPONSE

```
public static final short NO_T0_GETRESPONSE
```

This reason code indicates that during T=0 protocol, the CAD did not return a GET RESPONSE command in response to a <61xx> response status to send additional data. The outgoing transfer has been aborted. No more data or status can be sent to the CAD in this `APDU.process()` method.

T1_IFD_ABORT

```
public static final short T1_IFD_ABORT
```

This reason code indicates that during T=1 protocol, the CAD returned an ABORT S-Block command and aborted the data transfer. The incoming or outgoing transfer has been aborted. No more data can be received from the CAD. No more data or status can be sent to the CAD in this `APDU.process()` method.

Constructor Detail

APDUException

```
public APDUException(short reason)
```

Constructs an `APDUException`. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception.

Method Detail

throwIt

```
public static void throwIt(short reason)
```

Throws the JCRE owned instance of `APDUException` with the specified reason.

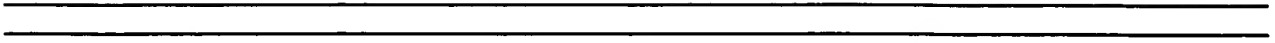
JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

reason - the reason for the exception.

Throws:

`APDUException` - always.



javacard.framework

Class Applet

```
java.lang.Object
|
+---javacard.framework.Applet
```

public abstract class Applet
extends Object

This abstract class defines an applet in Java Card.

The Applet class should be extended by any applet that is intended to be loaded onto, installed into and executed on a Java Card compliant smart card.

Example usage of Applet

```
public class MyApplet extends javacard.framework.Applet{
    static byte someByteArray[];

    public static void install( byte[] bArray, short bOffset, byte bLength ) throws ISOException {
        // make all my allocations here, so I do not run
        // out of memory later
        MyApplet theApplet = new MyApplet();

        // check incoming parameter
        byte bLen = bArray[bOffset];
        if ( bLen!=0 ) { someByteArray = new byte[bLen]; theApplet.register(); return; }
        else ISOException.throwIt(ISO7816.SW_FUNC_NOT_SUPPORTED);
    }

    public boolean select(){
        // selection initialization
        someByteArray[17] = 42; // set selection state
        return true;
    }

    public void process(APDU apdu) throws ISOException{
        byte[] buffer = apdu.getBuffer();
        // .. process the incoming data and reply
        if ( buffer[ISO7816.OFFSET_CLA] == (byte)0 ) {
            switch ( buffer[ISO7816.OFFSET_INS] ) {
                case ISO.INS_SELECT:
                    ...
                    // send response data to select command
                    short Le = apdu.setOutgoing();
                    // assume data containing response bytes in replyData[] array.
                    if ( Le < .. ) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH);
                    apdu.setOutgoingLength( (short)replyData.length );
                    apdu.sendBytesLong(replyData, (short) 0, (short)replyData.length);
                    break;
                case ...
            }
        }
    }
}
```


See Also:

`SystemException`, `JCSystem`

Constructor Summary

protected	Applet() Only this class's <code>install()</code> method should create the applet object.
-----------	---

Method Summary

void	deselect() Called by the JCRE to inform this currently selected applet that another (or the same) applet will be selected.
Shareable	getShareableInterfaceObject (AID clientAID, byte parameter) Called by the JCRE to obtain a shareable interface object from this server applet, on behalf of a request from a client applet.
static void	install (byte[] bArray, short bOffset, byte bLength) To create an instance of the Applet subclass, the JCRE will call this static method first.
abstract void	process (APDU apdu) Called by the JCRE to process an incoming APDU command.
protected void	register() This method is used by the applet to register this applet instance with the JCRE and to assign the Applet subclass AID bytes as its instance AID bytes.
protected void	register (byte[] bArray, short bOffset, byte bLength) This method is used by the applet to register this applet instance with the JCRE and assign the specified AID bytes as its instance AID bytes.
boolean	select() Called by the JCRE to inform this applet that it has been selected.
protected boolean	selectingApplet() This method is used by the applet <code>process()</code> method to distinguish the SELECT APDU command which selected this applet, from all other other SELECT APDU commands which may relate to file or internal applet state selection.

Methods inherited from class java.lang.Object
--

<code>equals</code>

Constructor Detail

Applet

`protected Applet()`

Only this class's `install()` method should create the applet object.

Method Detail

install

```
public static void install(byte[] bArray,
                           short bOffset,
                           byte bLength)
    throws IOException
```

To create an instance of the `Applet` subclass, the JCRE will call this static method first.

The applet should perform any necessary initializations and must call one of the `register()` methods. The installation is considered successful when the call to `register()` completes without an exception. The installation is deemed unsuccessful if the `install` method does not call a `register()` method, or if an exception is thrown from within the `install` method prior to the call to a `register()` method, or if the `register()` method throws an exception. If the installation is unsuccessful, the JCRE must perform all the necessary clean up when it receives control. Successful installation makes the applet instance capable of being selected via a `SELECT` APDU command.

Installation parameters are supplied in the byte array parameter and must be in a format defined by the applet. The `bArray` object is a global array. If the applet desires to preserve any of this data, it should copy the data into its own object.

`bArray` is zeroed by the JCRE after the return from the `install()` method.

References to the `bArray` object cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

The implementation of this method provided by `Applet` class throws an `ISOException` with reason code = `ISO7816.SW_FUNC_NOT_SUPPORTED`.

Note:

- *Exceptions thrown by this method after successful installation are caught by the JCRE and processed by the Installer.*

Parameters:

`bArray` - the array containing installation parameters.

`bOffset` - the starting offset in `bArray`.

`bLength` - the length in bytes of the parameter data in `bArray`. The maximum value of `bLength` is 32.

process

```
public abstract void process(APDU apdu)
                    throws ISOException
```

Called by the JCRE to process an incoming APDU command. An applet is expected to perform the action requested and return response data if any to the terminal.

Upon normal return from this method the JCRE sends the ISO 7816-4 defined success status (90 00) in APDU response. If this method throws an `ISOException` the JCRE sends the associated reason code as the response status instead.

The JCRE zeroes out the APDU buffer before receiving a new APDU command from the CAD. The five header bytes of the APDU command are available in `APDU buffer[0..4]` at the time this method is called.

The APDU object parameter is a temporary JCRE Entry Point Object. A temporary JCRE Entry Point Object can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

Notes:

- *APDU buffer[5..] is undefined and should not be read or written prior to invoking the `APDU.setIncomingAndReceive()` method if incoming data is expected. Altering the APDU buffer[5..] could corrupt incoming data.*

Parameters:

`apdu` - the incoming APDU object

Throws:

`ISOException` - with the response bytes per ISO 7816-4

See Also:

`APDU`

select

```
public boolean select()
```

Called by the JCRE to inform this applet that it has been selected.

It is called when a SELECT APDU command is received and before the applet is selected. SELECT APDU commands use instance AID bytes for applet selection. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

A subclass of `Applet` should override this method if it should perform any initialization that may be required to process APDU commands that may follow. This method returns a boolean to indicate that it is ready to accept incoming APDU commands via its `process()` method. If this method returns false, it indicates to the JCRE that this Applet declines to be selected.

The implementation of this method provided by `Applet` class returns true.

Returns:

true to indicate success, false otherwise.

deselect

```
public void deselect()
```

Called by the JCRE to inform this currently selected applet that another (or the same) applet will be selected. It is called when a SELECT APDU command is received by the JCRE. This method is invoked prior to another applets or this very applets `select()` method being invoked.

A subclass of `Applet` should override this method if it has any cleanup or bookkeeping work to be performed before another applet is selected.

The default implementation of this method provided by `Applet` class does nothing.

Notes:

- *Unchecked exceptions thrown by this method are caught by the JCRE but the applet is deselected.*
 - *Transient objects of `JCSysm.CLEAR_ON_DESELECT` clear event type are cleared to their default value by the JCRE after this method.*
 - *This method is NOT called on reset or power loss.*
-

getShareableInterfaceObject

```
public Shareable getShareableInterfaceObject(AID clientAID,  
                                              byte parameter)
```

Called by the JCRE to obtain a shareable interface object from this server applet, on behalf of a request from a client applet. This method executes in the applet context of this applet instance. The client applet initiated this request by calling the `JCSystem.getAppletShareableInterfaceObject()` method. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

- `clientAID` - the AID object of the client applet.
- `parameter` - optional parameter byte. The parameter byte may be used by the client to specify which shareable interface object is being requested.

Returns:

the shareable interface object or null. Note:

- *The clientAID parameter is a JCRE owned AID instance. JCRE owned instances of AID are permanent JCRE Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.*

See Also:

`JCSystem.getAppletShareableInterfaceObject(AID, byte)`

register

```
protected final void register()
                        throws SystemException
```

This method is used by the applet to register this applet instance with the JCRE and to assign the Applet subclass AID bytes as its instance AID bytes. One of the `register()` methods must be called from within `install()` to be registered with the JCRE. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Throws:

`SystemException` - with the following reason codes:

- `SystemException.ILLEGAL_AID` if the Applet subclass AID bytes are in use or if the applet instance has previously called one of the `register()` methods.
-

register

```
protected final void register(byte[] bArray,
                              short bOffset,
                              byte bLength)
                        throws SystemException
```

This method is used by the applet to register this applet instance with the JCRE and assign the specified AID bytes as its instance AID bytes. One of the `register()` methods must be called from within `install()` to be registered with the JCRE. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

- `bArray` - the byte array containing the AID bytes.
- `bOffset` - the start of AID bytes in `bArray`.
- `bLength` - the length of the AID bytes in `bArray`.

Throws:

APDUException - with the following reason codes:

SystemException - with the following reason code:

- `SystemException.ILLEGAL_VALUE` if the `bLength` parameter is less than 5 or greater than 16.
- `SystemException.ILLEGAL_AID` if the specified instance AID bytes are in use or if the RID portion of the AID bytes in the `bArray` parameter does not match the RID portion of the Applet subclass AID bytes or if the applet instance has previously called one of the `register()` methods.

selectingApplet

`protected final boolean selectingApplet()`

This method is used by the applet `process()` method to distinguish the SELECT APDU command which selected this applet, from all other other SELECT APDU commands which may relate to file or internal applet state selection.

Returns:

true if this applet is being selected.

javacard.framework

Class CardException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--javacard.framework.CardException
  
```

Direct Known Subclasses:

UserException

```
public class CardException
```

```
extends Exception
```

The `CardException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`. The `reason` field encapsulates exception cause identifier in Java Card. All Java Card checked Exception classes should extend `CardException`. This class also provides a resource-saving mechanism (`throwIt()` method) for using a JCRE owned instance of this class.

Constructor Summary

CardException(short reason)

Construct a `CardException` instance with the specified reason.

Method Summary

short	getReason() Get reason code
void	setReason (short reason) Set reason code
static void	throwIt (short reason) Throw the JCRE owned instance of <code>CardException</code> class with the specified reason.

Methods inherited from class java.lang.Object

equals

Constructor Detail

CardException

public **CardException**(short reason)

Construct a CardException instance with the specified reason. To conserve on resources, use the `throwIt()` method to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception

Method Detail

getReason

public short **getReason**()

Get reason code

Returns:

the reason for the exception

setReason

public void **setReason**(short reason)

Set reason code

Parameters:

reason - the reason for the exception

throwIt

public static void **throwIt**(short reason)
throws CardException

Throw the JCRE owned instance of CardException class with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1*

Specification for details.

Parameters:

reason - the reason for the exception

Throws:

CardException - always.

javacard.framework

Class CardRuntimeException

```

java.lang.Object
|
+--java.lang.Throwable
|   |
|   +--java.lang.Exception
|       |
|       +--java.lang.RuntimeException
|           |
|           +--javacard.framework.CardRuntimeException

```

Direct Known Subclasses:

APDUEException, CryptoException, ISOException, PINException, SystemException, TransactionException

```

public class CardRuntimeException
extends RuntimeException

```

The `CardRuntimeException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`. The `reason` field encapsulates exception cause identifier in Java Card. All Java Card unchecked Exception classes should extend `CardRuntimeException`. This class also provides a resource-saving mechanism (`throwIt()` method) for using a JCRE owned instance of this class.

Constructor Summary

CardRuntimeException (short reason)	
Construct a <code>CardRuntimeException</code> instance with the specified reason.	

Method Summary

short	getReason() Get reason code
void	setReason (short reason) Set reason code
static void	throwIt (short reason) Throw the JCRE owned instance of the <code>CardRuntimeException</code> class with the specified reason.

Methods inherited from class java.lang.Object

equals

Constructor Detail

CardRuntimeException

```
public CardRuntimeException(short reason)
```

Construct a CardRuntimeException instance with the specified reason. To conserve on resources, use `throwIt()` method to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception

Method Detail

getReason

```
public short getReason()
```

Get reason code

Returns:

the reason for the exception

setReason

```
public void setReason(short reason)
```

Set reason code

Parameters:

reason - the reason for the exception

throwIt

```
public static void throwIt(short reason)
    throws CardRuntimeException
```

Throw the JCRE owned instance of the `CardRuntimeException` class with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

reason - the reason for the exception

Throws:

`CardRuntimeException` - always.

javacard.framework

Interface ISO7816

public abstract interface **ISO7816**

ISO7816 encapsulates constants related to ISO 7816-3 and ISO 7816-4. ISO7816 interface contains only static fields.

The static fields with **SW_** prefixes define constants for the ISO 7816-4 defined response status word. The fields which use the **_00** suffix require the low order byte to be customized appropriately e.g (ISO7816.SW_CORRECT_LENGTH_00 + (0x0025 & 0xFF)).

The static fields with **OFFSET_** prefixes define constants to be used to index into the APDU buffer byte array to access ISO 7816-4 defined header information.

Field Summary

static byte	CLA_ISO7816 APDU command CLA : ISO 7816 = 0x00
static byte	INS_EXTERNAL_AUTHENTICATE APDU command INS : EXTERNAL AUTHENTICATE = 0x82
static byte	INS_SELECT APDU command INS : SELECT = 0xA4
static byte	OFFSET_CDATA APDU command data offset : CDATA = 5
static byte	OFFSET_CLA APDU header offset : CLA = 0
static byte	OFFSET_INS APDU header offset : INS = 1
static byte	OFFSET_LC APDU header offset : LC = 4
static byte	OFFSET_P1 APDU header offset : P1 = 2
static byte	OFFSET_P2 APDU header offset : P2 = 3
static short	SW_APPLET_SELECT_FAILED Response status : Applet selection failed = 0x6999;

static short	SW_BYTES_REMAINING_00 Response status : Response bytes remaining = 0x6100
static short	SW_CLA_NOT_SUPPORTED Response status : CLA value not supported = 0x6E00
static short	SW_COMMAND_NOT_ALLOWED Response status : Command not allowed (no current EF) = 0x6986
static short	SW_CONDITIONS_NOT_SATISFIED Response status : Conditions of use not satisfied = 0x6985
static short	SW_CORRECT_LENGTH_00 Response status : Correct Expected Length (Le) = 0x6C00
static short	SW_DATA_INVALID Response status : Data invalid = 0x6984
static short	SW_FILE_FULL Response status : Not enough memory space in the file = 0x6A84
static short	SW_FILE_INVALID Response status : File invalid = 0x6983
static short	SW_FILE_NOT_FOUND Response status : File not found = 0x6A82
static short	SW_FUNC_NOT_SUPPORTED Response status : Function not supported = 0x6A81
static short	SW_INCORRECT_P1P2 Response status : Incorrect parameters (P1,P2) = 0x6A86
static short	SW_INS_NOT_SUPPORTED Response status : INS value not supported = 0x6D00
static short	SW_NO_ERROR Response status : No Error = (short)0x9000
static short	SW_RECORD_NOT_FOUND Response status : Record not found = 0x6A83
static short	SW_SECURITY_STATUS_NOT_SATISFIED Response status : Security condition not satisfied = 0x6982
static short	SW_UNKNOWN Response status : No precise diagnosis = 0x6F00
static short	SW_WRONG_DATA Response status : Wrong data = 0x6A80
static short	SW_WRONG_LENGTH Response status : Wrong length = 0x6700

static short	SW_WRONG_P1P2 Response status : Incorrect parameters (P1,P2) = 0x6B00
--------------	---

Field Detail

SW_NO_ERROR

public static final short **SW_NO_ERROR**

Response status : No Error = (short)0x9000

SW_BYTES_REMAINING_00

public static final short **SW_BYTES_REMAINING_00**

Response status : Response bytes remaining = 0x6100

SW_WRONG_LENGTH

public static final short **SW_WRONG_LENGTH**

Response status : Wrong length = 0x6700

SW_SECURITY_STATUS_NOT_SATISFIED

public static final short **SW_SECURITY_STATUS_NOT_SATISFIED**

Response status : Security condition not satisfied = 0x6982

SW_FILE_INVALID

public static final short **SW_FILE_INVALID**

Response status : File invalid = 0x6983

SW_DATA_INVALID

public static final short **SW_DATA_INVALID**

Response status : Data invalid = 0x6984

SW_CONDITIONS_NOT_SATISFIED

public static final short **SW_CONDITIONS_NOT_SATISFIED**

Response status : Conditions of use not satisfied = 0x6985

SW_COMMAND_NOT_ALLOWED

public static final short **SW_COMMAND_NOT_ALLOWED**

Response status : Command not allowed (no current EF) = 0x6986

SW_APPLET_SELECT_FAILED

public static final short **SW_APPLET_SELECT_FAILED**

Response status : Applet selection failed = 0x6999;

SW_WRONG_DATA

public static final short **SW_WRONG_DATA**

Response status : Wrong data = 0x6A80

SW_FUNC_NOT_SUPPORTED

public static final short **SW_FUNC_NOT_SUPPORTED**

Response status : Function not supported = 0x6A81

SW_FILE_NOT_FOUND

public static final short **SW_FILE_NOT_FOUND**

Response status : File not found = 0x6A82

SW_RECORD_NOT_FOUND

public static final short **SW_RECORD_NOT_FOUND**

Response status : Record not found = 0x6A83

SW_INCORRECT_P1P2

public static final short **SW_INCORRECT_P1P2**

Response status : Incorrect parameters (P1,P2) = 0x6A86

SW_WRONG_P1P2

public static final short **SW_WRONG_P1P2**

Response status : Incorrect parameters (P1,P2) = 0x6B00

SW_CORRECT_LENGTH_00

public static final short **SW_CORRECT_LENGTH_00**

Response status : Correct Expected Length (Le) = 0x6C00

SW_INS_NOT_SUPPORTED

public static final short **SW_INS_NOT_SUPPORTED**

Response status : INS value not supported = 0x6D00

SW_CLA_NOT_SUPPORTED

public static final short **SW_CLA_NOT_SUPPORTED**

Response status : CLA value not supported = 0x6E00

SW_UNKNOWN

public static final short **SW_UNKNOWN**

Response status : No precise diagnosis = 0x6F00

SW_FILE_FULL

public static final short **SW_FILE_FULL**

Response status : Not enough memory space in the file = 0x6A84

OFFSET_CLA

public static final byte **OFFSET_CLA**

APDU header offset : CLA = 0

OFFSET_INS

public static final byte **OFFSET_INS**

APDU header offset : INS = 1

OFFSET_P1

public static final byte **OFFSET_P1**

APDU header offset : P1 = 2

OFFSET_P2

public static final byte **OFFSET_P2**

APDU header offset : P2 = 3

OFFSET_LC

public static final byte **OFFSET_LC**

APDU header offset : LC = 4

OFFSET_CDATA

public static final byte **OFFSET_CDATA**

APDU command data offset : CDATA = 5

CLA_ISO7816

public static final byte **CLA_ISO7816**

APDU command CLA : ISO 7816 = 0x00

INS_SELECT

public static final byte **INS_SELECT**

APDU command INS : SELECT = 0xA4

INS_EXTERNAL_AUTHENTICATE

public static final byte **INS_EXTERNAL_AUTHENTICATE**

APDU command INS : EXTERNAL AUTHENTICATE = 0x82

javacard.framework

Class ISOException

```

java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--javacard.framework.CardRuntimeException
                |
                +--javacard.framework.ISOException
  
```

```

public class ISOException
extends CardRuntimeException
  
```

ISOException class encapsulates an ISO 7816-4 response status word as its reason code.

The APDU class throws JCRE owned instances of ISOException.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Constructor Summary

ISOException (short sw)	Constructs an ISOException instance with the specified status word.
--------------------------------	---

Method Summary

static void	throwIt (short sw) Throws the JCRE owned instance of the ISOException class with the specified status word.
-------------	---

Methods inherited from class javacard.framework.CardRuntimeException

getReason, setReason

Methods inherited from class java.lang.Object**equals****Constructor Detail****ISOException**

```
public ISOException(short sw)
```

Constructs an ISOException instance with the specified status word. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

sw - the ISO 7816-4 defined status word

Method Detail**throwIt**

```
public static void throwIt(short sw)
```

Throws the JCRE owned instance of the ISOException class with the specified status word.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

sw - ISO 7816-4 defined status word

Throws:

ISOException - always.

javacard.framework

Class JCSystem

```
java.lang.Object
|
+--javacard.framework.JCSystem
```

```
public final class JCSystem
extends Object
```

The `JCSystem` class includes a collection of methods to control applet execution, resource management, atomic transaction management and inter-applet object sharing in Java Card. All methods in `JCSystem` class are static methods.

The `JCSystem` class also includes methods to control the persistence and transience of objects. The term *persistent* means that objects and their values persist from one CAD session to the next, indefinitely. Persistent object values are updated atomically using transactions.

The `makeTransient...Array()` methods can be used to create *transient* arrays with primitive data components. Transient array data is lost (in an undefined state, but the real data is unavailable) immediately upon power loss, and is reset to the default value at the occurrence of certain events such as card reset or deselect. Updates to the values of transient arrays are not atomic and are not affected by transactions.

The JCRE maintains an atomic transaction commit buffer which is initialized on card reset (or power on). When a transaction is in progress, the JCRE journals all updates to persistent data space into this buffer so that it can always guarantee, at commit time, that everything in the buffer is written or nothing at all is written. The `JCSystem` includes methods to control an atomic transaction. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:

`SystemException`, `TransactionException`, `Applet`

Field Summary

static byte	CLEAR_ON_DESELECT This event code indicates that the contents of the transient object are cleared to the default value on applet deselection event or in <code>CLEAR_ON_RESET</code> cases.
static byte	CLEAR_ON_RESET This event code indicates that the contents of the transient object are cleared to the default value on card reset (or power on) event.
static byte	NOT_A_TRANSIENT_OBJECT This event code indicates that the object is not transient.

Method Summary	
static void	abortTransaction() Aborts the atomic transaction.
static void	beginTransaction() Begins an atomic transaction.
static void	commitTransaction() Commits an atomic transaction.
static AID	getAID() Returns the JCRE owned instance of the AID object associated with the current applet context.
static Shareable	getAppletShareableInterfaceObject (AID serverAID, byte parameter) This method is called by a client applet to get a server applet's shareable interface object.
static short	getMaxCommitCapacity() Returns the total number of bytes in the commit buffer.
static AID	getPreviousContextAID() This method is called to obtain the JCRE owned instance of the AID object associated with the previously active applet context.
static byte	getTransactionDepth() Returns the current transaction nesting depth level.
static short	getUnusedCommitCapacity() Returns the number of bytes left in the commit buffer.
static short	getVersion() Returns the current major and minor version of the Java Card API.
static byte	isTransient (Object theObj) Used to check if the specified object is transient.
static AID	lookupAID (byte[] buffer, short offset, byte length) Returns the JCRE owned instance of the AID object, if any, encapsulating the specified AID bytes in the buffer parameter if there exists a successfully installed applet on the card whose instance AID exactly matches that of the specified AID bytes.
static boolean[]	makeTransientBooleanArray (short length, byte event) Create a transient boolean array with the specified array length.
static byte[]	makeTransientByteArray (short length, byte event) Create a transient byte array with the specified array length.

static Object[]	makeTransientObjectArray (short length, byte event) Create a transient array of Object with the specified array length.
static short[]	makeTransientShortArray (short length, byte event) Create a transient short array with the specified array length.

Methods inherited from class java.lang.Object

equals

Field Detail

NOT_A_TRANSIENT_OBJECT

```
public static final byte NOT_A_TRANSIENT_OBJECT
```

This event code indicates that the object is not transient.

CLEAR_ON_RESET

```
public static final byte CLEAR_ON_RESET
```

This event code indicates that the contents of the transient object are cleared to the default value on card reset (or power on) event.

CLEAR_ON_DESELECT

```
public static final byte CLEAR_ON_DESELECT
```

This event code indicates that the contents of the transient object are cleared to the default value on applet deselection event or in CLEAR_ON_RESET cases.

Notes:

- CLEAR_ON_DESELECT *transient objects can be accessed only when the applet which created the object is the currently the selected applet.*
- The JCRE will throw a SecurityException if a CLEAR_ON_DESELECT transient object is accessed when the currently selected applet is not the applet which created the object.

Method Detail

isTransient

```
public static byte isTransient(Object theObj)
```

Used to check if the specified object is transient.

Notes:

This method returns NOT_A_TRANSIENT_OBJECT if the specified object is null or is not an array type.

Parameters:

theObj - the object being queried.

Returns:

NOT_A_TRANSIENT_OBJECT, CLEAR_ON_RESET, or CLEAR_ON_DESELECT.

See Also:

makeTransientBooleanArray(short, byte),
makeTransientByteArray(short, byte),
makeTransientShortArray(short, byte),
makeTransientObjectArray(short, byte)

makeTransientBooleanArray

```
public static boolean[] makeTransientBooleanArray(short length,
                                                    byte event)
                                                    throws SystemException
```

Create a transient boolean array with the specified array length.

Parameters:

length - the length of the boolean array.

event - the CLEAR_ON... event which causes the array elements to be cleared.

Throws:

SystemException - with the following reason codes:

- SystemException.ILLEGAL_VALUE if event is not a valid event code.
 - SystemException.NO_TRANSIENT_SPACE if sufficient transient space is not available.
 - SystemException.ILLEGAL_TRANSIENT if the current applet context is not the currently selected applet context and CLEAR_ON_DESELECT is specified.
-

makeTransientByteArray

```
public static byte[] makeTransientByteArray(short length,
                                              byte event)
                                              throws SystemException
```

Create a transient byte array with the specified array length.

Parameters:

length - the length of the byte array.

event - the CLEAR_ON... event which causes the array elements to be cleared.

Throws:

SystemException - with the following reason codes:

- SystemException.ILLEGAL_VALUE if event is not a valid event code.
 - SystemException.NO_TRANSIENT_SPACE if sufficient transient space is not available.
 - SystemException.ILLEGAL_TRANSIENT if the current applet context is not the currently selected applet context and CLEAR_ON_DESELECT is specified.
-

makeTransientShortArray

```
public static short[] makeTransientShortArray(short length,  
                                              byte event)  
    throws SystemException
```

Create a transient short array with the specified array length.

Parameters:

length - the length of the short array.

event - the CLEAR_ON... event which causes the array elements to be cleared.

Throws:

SystemException - with the following reason codes:

- SystemException.ILLEGAL_VALUE if event is not a valid event code.
 - SystemException.NO_TRANSIENT_SPACE if sufficient transient space is not available.
 - SystemException.ILLEGAL_TRANSIENT if the current applet context is not the currently selected applet context and CLEAR_ON_DESELECT is specified.
-

makeTransientObjectArray

```
public static Object[] makeTransientObjectArray(short length,  
                                                byte event)  
    throws SystemException
```

Create a transient array of Object with the specified array length.

Parameters:

length - the length of the Object array.

event - the CLEAR_ON... event which causes the array elements to be cleared.

Throws:

SystemException - with the following reason codes:

- SystemException.ILLEGAL_VALUE if event is not a valid event code.
- SystemException.NO_TRANSIENT_SPACE if sufficient transient space is not available.
- SystemException.ILLEGAL_TRANSIENT if the current applet context is not the currently selected applet context and CLEAR_ON_DESELECT is specified.

getVersion

```
public static short getVersion()
```

Returns the current major and minor version of the Java Card API.

Returns:

version number as byte.byte (major.minor)

getAID

```
public static AID getAID()
```

Returns the JCRE owned instance of the AID object associated with the current applet context. Returns null if the `Applet.register()` method has not yet been invoked.

JCRE owned instances of AID are permanent JCRE Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Returns:

the AID object.

lookupAID

```
public static AID lookupAID(byte[] buffer,  
                           short offset,  
                           byte length)
```

Returns the JCRE owned instance of the AID object, if any, encapsulating the specified AID bytes in the `buffer` parameter if there exists a successfully installed applet on the card whose instance AID exactly matches that of the specified AID bytes.

JCRE owned instances of AID are permanent JCRE Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

`buffer` - byte array containing the AID bytes.

`offset` - offset within buffer where AID bytes begin.

`length` - length of AID bytes in buffer.

Returns:

the AID object, if any; null otherwise. A VM exception is thrown if `buffer` is null, or if `offset` or `length` are out of range.

beginTransaction

```
public static void beginTransaction()
                    throws TransactionException
```

Begins an atomic transaction. If a transaction is already in progress (`transactionDepth != 0`), a `TransactionException` is thrown.

Throws:

`TransactionException` - with the following reason codes:

- `TransactionException.IN_PROGRESS` if a transaction is already in progress.

See Also:

`commitTransaction()`, `abortTransaction()`

abortTransaction

```
public static void abortTransaction()
                    throws TransactionException
```

Aborts the atomic transaction. The contents of the commit buffer is discarded.

Notes:

- *Do not call this method from within a transaction which creates new objects because the JCRE may not recover the heap space used by the new object instances.*
- *The JCRE ensures that any variable of reference type which references an object instantiated from within this aborted transaction is equivalent to a null reference.*

Throws:

`TransactionException` - with the following reason codes:

- `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

See Also:

`beginTransaction()`, `commitTransaction()`

commitTransaction

```
public static void commitTransaction()
                    throws TransactionException
```

Commits an atomic transaction. The contents of commit buffer is atomically committed. If a transaction is not in progress (`transactionDepth == 0`) then a `TransactionException` is thrown.

Throws:

`TransactionException` - with the following reason codes:

- `TransactionException.NOT_IN_PROGRESS` if a transaction is not in progress.

See Also:

`beginTransaction()`, `abortTransaction()`

getTransactionDepth

```
public static byte getTransactionDepth()
```

Returns the current transaction nesting depth level. At present, only 1 transaction can be in progress at a time.

Returns:

1 if transaction in progress, 0 if not.

getUnusedCommitCapacity

```
public static short getUnusedCommitCapacity()
```

Returns the number of bytes left in the commit buffer.

Returns:

the number of bytes left in the commit buffer

See Also:

`getMaxCommitCapacity()`

getMaxCommitCapacity

```
public static short getMaxCommitCapacity()
```

Returns the total number of bytes in the commit buffer. This is approximately the maximum number of bytes of persistent data which can be modified during a transaction. However, the transaction subsystem requires additional bytes of overhead data to be included in the commit buffer, and this depends on the number of fields modified and the implementation of the transaction subsystem. The application cannot determine the actual maximum amount of data which can be modified during a transaction without taking these overhead bytes into consideration.

Returns:

the total number of bytes in the commit buffer

See Also:

`getUnusedCommitCapacity()`

getPreviousContextAID

```
public static AID getPreviousContextAID()
```

This method is called to obtain the JCRE owned instance of the AID object associated with the previously active applet context. This method is typically used by a server applet, while executing a shareable interface method to determine the identity of its client and thereby control access privileges.

JCRE owned instances of AID are permanent JCRE Entry Point Objects and can be accessed from any applet context. References to these permanent objects can be stored and re-used.

See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Returns:

the AID object of the previous context, or null if JCRE.

getAppletShareableInterfaceObject

```
public static Shareable getAppletShareableInterfaceObject(AID serverAID,  
                                                         byte parameter)
```

This method is called by a client applet to get a server applet's shareable interface object.

This method returns null if the `Applet.register()` has not yet been invoked or if the server does not exist or if the server returns null.

Parameters:

`serverAID` - the AID of the server applet.

`parameter` - optional parameter data.

Returns:

the shareable interface object or null.

See Also:

`Applet.getShareableInterfaceObject(AID, byte)`

javacard.framework

Class OwnerPIN

```
java.lang.Object
|
+---javacard.framework.OwnerPIN
```

```
public class OwnerPIN
extends Object
implements PIN
```

This class represents an Owner PIN. It implements Personal Identification Number functionality as defined in the PIN interface. It provides the ability to update the PIN and thus owner functionality.

The implementation of this class must protect against attacks based on program flow prediction. Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated during PIN presentation.

If an implementation of this class creates transient arrays, it must ensure that they are CLEAR_ON_RESET transient objects.

The protected methods `getValidatedFlag` and `setValidatedFlag` allow a subclass of this class to optimize the storage for the validated boolean state.

Some methods of instances of this class are only suitable for sharing when there exists a trust relationship among the applets. A typical shared usage would use a proxy PIN interface which implements both the PIN interface and the `Shareable` interface.

Any of the methods of the `OwnerPIN` may be called with a transaction in progress. None of the methods of `OwnerPIN` class initiate or alter the state of the transaction if one is in progress.

See Also:

`PINException`, `PIN`, `Shareable`, `JCSys`

Constructor Summary

OwnerPIN (byte tryLimit, byte maxPINSize) Constructor.	
--	--

Method Summary

boolean	check (byte[] pin, short offset, byte length) Compares pin against the PIN value.
byte	getTriesRemaining () Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.
protected boolean	getValidatedFlag () This protected method returns the validated flag.
boolean	isValidated () Returns true if a valid PIN has been presented since the last card reset or last call to <code>reset()</code> .
void	reset () If the validated flag is set, this method resets it.
void	resetAndUnblock () This method resets the validated flag and resets the PIN try counter to the value of the PIN try limit.
protected void	setValidatedFlag (boolean value) This protected method sets the value of the validated flag.
void	update (byte[] pin, short offset, byte length) This method sets a new value for the PIN and resets the PIN try counter to the value of the PIN try limit.

Methods inherited from class java.lang.Object

`equals`

Constructor Detail

OwnerPIN

```
public OwnerPIN(byte tryLimit,
                byte maxPINSize)
    throws PINException
```

Constructor. Allocates a new PIN instance with validated flag set to false.

Parameters:

`tryLimit` - the maximum number of times an incorrect PIN can be presented.
`maxPINSize` - the maximum allowed PIN size. `maxPINSize` must be ≥ 1 .

Throws:

PINException - with the following reason codes:

- PINException.ILLEGAL_VALUE if maxPINSize parameter is less than 1.

Method Detail

getValidatedFlag

protected boolean **getValidatedFlag**()

This protected method returns the validated flag. This method is intended for subclass of this OwnerPIN to access or override the internal PIN state of the OwnerPIN.

Returns:

the boolean state of the PIN validated flag.

setValidatedFlag

protected void **setValidatedFlag**(boolean value)

This protected method sets the value of the validated flag. This method is intended for subclass of this OwnerPIN to control or override the internal PIN state of the OwnerPIN.

Parameters:

value - the new value for the validated flag.

getTriesRemaining

public byte **getTriesRemaining**()

Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.

Specified by:

getTriesRemaining in interface PIN

Returns:

the number of times remaining

check

```
public boolean check(byte[] pin,
                    short offset,
                    byte length)
    throws ArrayIndexOutOfBoundsException,
           NullPointerException
```

Compares `pin` against the PIN value. If they match and the PIN is not blocked, it sets the validated flag and resets the try counter to its maximum. If it does not match, it decrements the try counter and, if the counter has reached zero, blocks the PIN. Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated.

Notes:

- *If `NullPointerException` or `ArrayIndexOutOfBoundsException` is thrown, the validated flag must be set to false, the try counter must be decremented and, the PIN blocked if the counter reaches zero.*
- *If `offset` or `length` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `offset+length` is greater than `pin.length`, the length of the pin array, an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `pin` parameter is null a `NullPointerException` exception is thrown.*

Specified by:

check in interface PIN

Parameters:

`pin` - the byte array containing the PIN value being checked

`offset` - the starting offset in the pin array

`length` - the length of pin.

Returns:

true if the PIN value matches; false otherwise

Throws:

`ArrayIndexOutOfBoundsException` - - if the check operation would cause access of data outside array bounds.

`NullPointerException` - - if `pin` is null

isValidated

```
public boolean isValidated()
```

Returns true if a valid PIN has been presented since the last card reset or last call to `reset()`.

Specified by:

`isValidated` in interface PIN

Returns:

true if validated; false otherwise

reset

```
public void reset()
```

If the validated flag is set, this method resets it. If the validated flag is not set, this method does nothing.

Specified by:

`reset` in interface PIN

update

```
public void update(byte[] pin,  
                  short offset,  
                  byte length)  
    throws PINException
```

This method sets a new value for the PIN and resets the PIN try counter to the value of the PIN try limit. It also resets the validated flag.

This method copies the input pin parameter into an internal representation. If a transaction is in progress, the new pin and try counter update must be conditional i.e the copy operation must use the transaction facility.

Parameters:

- pin - the byte array containing the new PIN value
- offset - the starting offset in the pin array
- length - the length of the new PIN.

Throws:

PINException - with the following reason codes:

- PINException.ILLEGAL_VALUE if length is greater than configured maximum PIN size.

See Also:

JCSystem.beginTransaction()

resetAndUnblock

```
public void resetAndUnblock()
```

This method resets the validated flag and resets the PIN try counter to the value of the PIN try limit. This method is used by the owner to re-enable the blocked PIN.

javacard.framework

Interface PIN

All Known Implementing Classes:
OwnerPIN

public abstract interface **PIN**

This interface represents a PIN. An implementation must maintain these internal values:

- PIN value
- try limit, the maximum number of times an incorrect PIN can be presented before the PIN is blocked. When the PIN is blocked, it cannot be validated even on valid PIN presentation.
- max PIN size, the maximum length of PIN allowed
- try counter, the remaining number of times an incorrect PIN presentation is permitted before the PIN becomes blocked.
- validated flag, true if a valid PIN has been presented. This flag is reset on every card reset.

This interface does not make any assumptions about where the data for the PIN value comparison is stored.

An owner implementation of this interface must provide a way to initialize/update the PIN value. The owner implementation of the interface must protect against attacks based on program flow prediction. Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated during PIN presentation.

A typical card global PIN usage will combine an instance of OwnerPIN class and a Proxy PIN interface which implements both the PIN and the Shareable interfaces. The OwnerPIN instance would be manipulated only by the owner who has update privilege. All others would access the global PIN functionality via the proxy PIN interface.

See Also:

OwnerPIN, Shareable

Method Summary

boolean	check (byte[] pin, short offset, byte length) Compares pin against the PIN value.
byte	getTriesRemaining () Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.
boolean	isValidated () Returns true if a valid PIN value has been presented since the last card reset or last call to reset().
void	reset () If the validated flag is set, this method resets it.

Method Detail

getTriesRemaining

```
public byte getTriesRemaining()
```

Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.

Returns:

the number of times remaining

check

```
public boolean check(byte[] pin,
                    short offset,
                    byte length)
    throws ArrayIndexOutOfBoundsException,
           NullPointerException
```

Compares pin against the PIN value. If they match and the PIN is not blocked, it sets the validated flag and resets the try counter to its maximum. If it does not match, it decrements the try counter and, if the counter has reached zero, blocks the PIN. Even if a transaction is in progress, internal state such as the try counter, the validated flag and the blocking state must not be conditionally updated.

Notes:

- *If NullPointerException or ArrayIndexOutOfBoundsException is thrown, the validated flag must be set to false, the try counter must be decremented and, the PIN blocked if the counter reaches zero.*
If offset or length parameter is negative an ArrayIndexOutOfBoundsException

exception is thrown.

If offset+length is greater than pin.length, the length of the pin array, an `ArrayIndexOutOfBoundsException` exception is thrown.

- *If pin parameter is null a `NullPointerException` exception is thrown.*

Parameters:

pin - the byte array containing the PIN value being checked

offset - the starting offset in the pin array

length - the length of pin.

Returns:

true if the PIN value matches; false otherwise

Throws:

`ArrayIndexOutOfBoundsException` - - if the check operation would cause access of data outside array bounds.

`NullPointerException` - - if pin is null

isValidated

```
public boolean isValidated()
```

Returns true if a valid PIN value has been presented since the last card reset or last call to `reset()`.

Returns:

true if validated; false otherwise

reset

```
public void reset()
```

If the validated flag is set, this method resets it. If the validated flag is not set, this method does nothing.

javacard.framework

Class PINException

```

java.lang.Object
|
+---java.lang.Throwable
      |
      +---java.lang.Exception
            |
            +---java.lang.RuntimeException
                  |
                  +---javacard.framework.CardRuntimeException
                        |
                        +---javacard.framework.PINException
  
```

public class **PINException**
 extends CardRuntimeException

PINException represents a OwnerPIN class access-related exception.

The OwnerPIN class throws JCRE owned instances of PINException.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:

OwnerPIN

Field Summary

static short	ILLEGAL_VALUE This reason code is used to indicate that one or more input parameters is out of allowed bounds.
--------------	--

Constructor Summary

PINException (short reason) Constructs a PINException.	
--	--

Method Summary

static void	throwIt (short reason) Throws the JCRE owned instance of <code>PINException</code> with the specified reason.
-------------	---

Methods inherited from class `javacard.framework.CardRuntimeException`

<code>getReason</code> , <code>setReason</code>

Methods inherited from class `java.lang.Object`

<code>equals</code>

Field Detail

ILLEGAL_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

Constructor Detail

PINException

```
public PINException(short reason)
```

Constructs a `PINException`. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception.

Method Detail

throwIt

public static void **throwIt**(short reason)

Throws the JCRE owned instance of `PINException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

reason - the reason for the exception.

Throws:

`PINException` - always.

javacard.framework

Interface Shareable

public abstract interface Shareable

The Shareable interface serves to identify all shared objects. Any object that needs to be shared through the applet firewall must directly or indirectly implement this interface. Only those methods specified in a shareable interface are available through the firewall. Implementation classes can implement any number of shareable interfaces and can extend other shareable implementation classes.

javacard.framework

Class SystemException

```
java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException
|
+--javacard.framework.CardRuntimeException
|
+--javacard.framework.SystemException
```

```
public class SystemException
extends CardRuntimeException
```

SystemException represents a JCSys`tem` class related exception. It is also thrown by the `javacard.framework.Applet.register()` methods and by the AID class constructor.

These API classes throw JCRE owned instances of **SystemException**.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:

JCSys`tem`, Applet, AID

Field Summary	
static short	ILLEGAL_AID This reason code is used by the <code>javacard.framework.Applet.register()</code> method to indicate that the input AID parameter is not a legal AID value.
static short	ILLEGAL_TRANSIENT This reason code is used to indicate that the request to create a transient object is not allowed in the current applet context.
static short	ILLEGAL_VALUE This reason code is used to indicate that one or more input parameters is out of allowed bounds.
static short	NO_RESOURCE This reason code is used to indicate that there is insufficient resource in the Card for the request.
static short	NO_TRANSIENT_SPACE This reason code is used by the <code>makeTransient..()</code> methods to indicate that no room is available in volatile memory for the requested object.

Constructor Summary	
SystemException (short reason)	
Constructs a SystemException.	

Method Summary	
static void	throwIt (short reason) Throws the JCRC owned instance of <code>SystemException</code> with the specified reason.

Methods inherited from class <code>javacard.framework.CardRuntimeExcepion</code>	
<code>getReason, setReason</code>	

Methods inherited from class java.lang.Object

equals

Field Detail

ILLEGAL_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

NO_TRANSIENT_SPACE

```
public static final short NO_TRANSIENT_SPACE
```

This reason code is used by the `makeTransient..()` methods to indicate that no room is available in volatile memory for the requested object.

ILLEGAL_TRANSIENT

```
public static final short ILLEGAL_TRANSIENT
```

This reason code is used to indicate that the request to create a transient object is not allowed in the current applet context. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

ILLEGAL_AID

```
public static final short ILLEGAL_AID
```

This reason code is used by the `javacard.framework.Applet.register()` method to indicate that the input AID parameter is not a legal AID value.

NO_RESOURCE

```
public static final short NO_RESOURCE
```

This reason code is used to indicate that there is insufficient resource in the Card for the request.

For example, the Java Card Virtual Machine may throw this exception reason when there is insufficient heap space to create a new instance.

Constructor Detail

SystemException

`public SystemException(short reason)`

Constructs a SystemException. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

`reason` - the reason for the exception.

Method Detail

throwIt

`public static void throwIt(short reason)`

Throws the JCRE owned instance of `SystemException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

`reason` - the reason for the exception.

Throws:

`SystemException` - always.

javacard.framework

Class TransactionException

```
java.lang.Object
|
+--java.lang.Throwable
    |
    +--java.lang.Exception
        |
        +--java.lang.RuntimeException
            |
            +--javacard.framework.CardRuntimeException
                |
                +--javacard.framework.TransactionException
```

```
public class TransactionException
extends CardRuntimeException
```

TransactionException represents an exception in the transaction subsystem. The methods referred to in this class are in the JCSYSTEM class.

The JCSYSTEM class and the transaction facility throw JCRE owned instances of TransactionException.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

See Also:

JCSYSTEM

Field Summary

static short	BUFFER_FULL This reason code is used during a transaction to indicate that the commit buffer is full.
static short	IN_PROGRESS This reason code is used by the <code>beginTransaction</code> method to indicate a transaction is already in progress.
static short	INTERNAL_FAILURE This reason code is used during a transaction to indicate an internal JCRE problem (fatal error).
static short	NOT_IN_PROGRESS This reason code is used by the <code>abortTransaction</code> and <code>commitTransaction</code> methods when a transaction is not in progress.

Constructor Summary

TransactionException (short reason)	
Constructs a TransactionException with the specified reason.	

Method Summary

static void	throwIt (short reason)
	Throws the JCRE owned instance of TransactionException with the specified reason.

Methods inherited from class `javacard.framework.CardRuntimeException`

`getReason`, `setReason`

Methods inherited from class `java.lang.Object`

`equals`

Field Detail

IN_PROGRESS

```
public static final short IN_PROGRESS
```

This reason code is used by the `beginTransaction` method to indicate a transaction is already in progress.

NOT_IN_PROGRESS

```
public static final short NOT_IN_PROGRESS
```

This reason code is used by the `abortTransaction` and `commitTransaction` methods when a transaction is not in progress.

BUFFER_FULL

```
public static final short BUFFER_FULL
```

This reason code is used during a transaction to indicate that the commit buffer is full.

INTERNAL_FAILURE

```
public static final short INTERNAL_FAILURE
```

This reason code is used during a transaction to indicate an internal JCRE problem (fatal error).

Constructor Detail

TransactionException

```
public TransactionException(short reason)
```

Constructs a `TransactionException` with the specified reason. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Method Detail

throwIt

public static void **throwIt**(short reason)

Throws the JCRE owned instance of `TransactionException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Throws:

`TransactionException` - always.

javacard.framework

Class UserException

```

java.lang.Object
|
+--java.lang.Throwable
|   |
|   +--java.lang.Exception
|       |
|       +--javacard.framework.CardException
|           |
|           +--javacard.framework.UserException

```

```

public class UserException
extends CardException

```

UserException represents a User exception. This class also provides a resource-saving mechanism (the `throwIt()` method) for user exceptions by using a JCRE owned instance.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Constructor Summary

UserException()	Constructs a UserException with reason = 0.
UserException(short reason)	Constructs a UserException with the specified reason.

Method Summary

static void	throwIt(short reason) Throws the JCRE owned instance of UserException with the specified reason.
-------------	--

Methods inherited from class javacard.framework.CardException

getReason, setReason

Methods inherited from class java.lang.Object

equals

Constructor Detail

UserException

`public UserException()`

Constructs a `UserException` with reason = 0. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

UserException

`public UserException(short reason)`

Constructs a `UserException` with the specified reason. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

reason - the reason for the exception.

Method Detail

throwIt

`public static void throwIt(short reason)`
throws `UserException`

Throws the JCRE owned instance of `UserException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

reason - the reason for the exception.

Throws:

`UserException` - always.



javacard.framework

Class Util

```
java.lang.Object
|
+--javacard.framework.Util
```

```
public class Util
extends Object
```

The `Util` class contains common utility functions. Some of the methods may be implemented as native functions for performance reasons. All methods in `Util`, class are static methods.

Some methods of `Util` namely `arrayCopy()`, `arrayCopyNonAtomic()`, `arrayFillNonAtomic()` and `setShort()`, refer to the persistence of array objects. The term *persistent* means that arrays and their values persist from one CAD session to the next, indefinitely. The `JCSys`tem class is used to control the persistence and transience of objects.

See Also:

`JCSys`tem

Method Summary

static byte	arrayCompare (byte[] src, short srcOff, byte[] dest, short destOff, short length) Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right.
static short	arrayCopy (byte[] src, short srcOff, byte[] dest, short destOff, short length) Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.
static short	arrayCopyNonAtomic (byte[] src, short srcOff, byte[] dest, short destOff, short length) Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically).
static short	arrayFillNonAtomic (byte[] bArray, short bOff, short bLen, byte bValue) Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value.
static short	getShort (byte[] bArray, short bOff) Concatenates two bytes in a byte array to form a short value.
static short	makeShort (byte b1, byte b2) Concatenates the two parameter bytes to form a short value.
static short	setShort (byte[] bArray, short bOff, short sValue) Deposits the short value as two successive bytes at the specified offset in the byte array.

Methods inherited from class java.lang.Object

equals

Method Detail

arrayCopy

```
public static final short arrayCopy(byte[] src,
                                   short srcOff,
                                   byte[] dest,
                                   short destOff,
```

short length)
throws `ArrayIndexOutOfBoundsException`,
`NullPointerException`,
`TransactionException`

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

Notes:

- *If `srcOff` or `destOff` or `length` parameter is negative an `ArrayIndexOutOfBoundsException` exception is thrown.*
- *If `srcOff+length` is greater than `src.length`, the length of the `src` array a `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `destOff+length` is greater than `dest.length`, the length of the `dest` array an `ArrayIndexOutOfBoundsException` exception is thrown and no copy is performed.*
- *If `src` or `dest` parameter is null a `NullPointerException` exception is thrown.*
- *If the `src` and `dest` arguments refer to the same array object, then the copying is performed as if the components at positions `srcOff` through `srcOff+length-1` were first copied to a temporary array with length components and then the contents of the temporary array were copied into positions `destOff` through `destOff+length-1` of the argument array.*
- *If the destination array is persistent, the entire copy is performed atomically.*
- *The copy operation is subject to atomic commit capacity limitations. If the commit capacity is exceeded, no copy is performed and a `TransactionException` exception is thrown.*

Parameters:

`src` - source byte array.
`srcOff` - offset within source byte array to start copy from.
`dest` - destination byte array.
`destOff` - offset within destination byte array to start copy into.
`length` - byte length to be copied.

Returns:

`destOff+length`

Throws:

`ArrayIndexOutOfBoundsException` - - if copying would cause access of data outside array bounds.
`NullPointerException` - - if either `src` or `dest` is null.
`TransactionException` - - if copying would cause the commit capacity to be exceeded.

See Also:

`JCSystem.getUnusedCommitCapacity()`

arrayCopyNonAtomic

```
public static final short arrayCopyNonAtomic(byte[] src,  
                                              short srcOff,  
                                              byte[] dest,
```



```

        short destOff,
        short length)
    throws ArrayIndexOutOfBoundsException,
           NullPointerException

```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically).

This method does not use the transaction facility during the copy operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the destination array can be left in a partially modified state in the event of a power loss in the middle of the copy operation.

Notes:

- *If srcOff or destOff or length parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If srcOff+length is greater than src.length, the length of the src array a ArrayIndexOutOfBoundsException exception is thrown and no copy is performed.*
- *If destOff+length is greater than dest.length, the length of the dest array an ArrayIndexOutOfBoundsException exception is thrown and no copy is performed.*
- *If src or dest parameter is null a NullPointerException exception is thrown.*
- *If the src and dest arguments refer to the same array object, then the copying is performed as if the components at positions srcOff through srcOff+length-1 were first copied to a temporary array with length components and then the contents of the temporary array were copied into positions destOff through destOff+length-1 of the argument array.*
- *If power is lost during the copy operation and the destination array is persistent, a partially changed destination array could result.*
- *The copy length parameter is not constrained by the atomic commit capacity limitations.*

Parameters:

src - source byte array.
 srcOff - offset within source byte array to start copy from.
 dest - destination byte array.
 destOff - offset within destination byte array to start copy into.
 length - byte length to be copied.

Returns:

destOff+length

Throws:

ArrayIndexOutOfBoundsException - - if copying would cause access of data outside array bounds.
 NullPointerException - - if either src or dest is null.

See Also:

JCSystem.getUnusedCommitCapacity()

arrayFillNonAtomic

```
public static final short arrayFillNonAtomic(byte[] bArray,
                                             short bOff,
                                             short bLen,
                                             byte bValue)
    throws ArrayIndexOutOfBoundsException,
           NullPointerException
```

Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value.

This method does not use the transaction facility during the fill operation even if a transaction is in progress. Thus, this method is suitable for use only when the contents of the byte array can be left in a partially filled state in the event of a power loss in the middle of the fill operation.

Notes:

- *If bOff or bLen parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If bOff+bLen is greater than bArray.length, the length of the bArray array an ArrayIndexOutOfBoundsException exception is thrown.*
- *If bArray parameter is null a NullPointerException exception is thrown.*
- *If power is lost during the copy operation and the byte array is persistent, a partially changed byte array could result.*
- *The bLen parameter is not constrained by the atomic commit capacity limitations.*

Parameters:

bArray - the byte array.
 bOff - offset within byte array to start filling bValue into.
 bLen - byte length to be filled.
 bValue - the value to fill the byte array with.

Returns:

bOff+bLen

Throws:

ArrayIndexOutOfBoundsException - - if the fill operation would cause access of data outside array bounds.

NullPointerException - - if bArray is null

See Also:

JCSystem.getUnusedCommitCapacity()

arrayCompare

```
public static final byte arrayCompare(byte[] src,
                                     short srcOff,
                                     byte[] dest,
                                     short destOff,
                                     short length)
    throws ArrayIndexOutOfBoundsException,
           NullPointerException
```

Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right. Returns the ternary result of the comparison : less than(-1), equal(0) or greater than(1).

Notes:

- *If srcOff or destOff or length parameter is negative an ArrayIndexOutOfBoundsException exception is thrown.*
- *If srcOff+length is greater than src.length, the length of the src array a ArrayIndexOutOfBoundsException exception is thrown.*
- *If destOff+length is greater than dest.length, the length of the dest array an ArrayIndexOutOfBoundsException exception is thrown.*
- *If src or dest parameter is null a NullPointerException exception is thrown.*

Parameters:

src - source byte array.
 srcOff - offset within source byte array to start compare.
 dest - destination byte array.
 destOff - offset within destination byte array to start compare.
 length - byte length to be compared.

Returns:

the result of the comparison as follows:

- 0 if identical
- -1 if the first miscomparing byte in source array is less than that in destination array,
- 1 if the first miscomparing byte in source array is greater that that in destination array.

Throws:

ArrayIndexOutOfBoundsException - - if comparing all bytes would cause access of data outside array bounds.
 NullPointerException - - if either src or dest is null.

makeShort

```
public static final short makeShort(byte b1,
                                   byte b2)
```

Concatenates the two parameter bytes to form a short value.

Parameters:

b1 - the first byte (high order byte).
 b2 - the second byte (low order byte).

Returns:

the short value - the concatenated result

getShort

```
public static final short getShort(byte[] bArray,  
                                     short bOff)
```

Concatenates two bytes in a byte array to form a short value.

Parameters:

bArray - byte array.

bOff - offset within byte array containing first byte (the high order byte).

Returns:

the short value - the concatenated result

setShort

```
public static final short setShort(byte[] bArray,  
                                     short bOff,  
                                     short sValue)  
    throws TransactionException
```

Deposits the short value as two successive bytes at the specified offset in the byte array.

Parameters:

bArray - byte array.

bOff - offset within byte array to deposit the first byte (the high order byte).

sValue - the short value to set into array.

Returns:

bOff+2

Note:

- *If the byte array is persistent, this operation is performed atomically. If the commit capacity is exceeded, no operation is performed and a TransactionException exception is thrown.*

Throws:

TransactionException - - if the operation would cause the commit capacity to be exceeded.

See Also:

JCSystem.getUnusedCommitCapacity()

Package javacard.security

Provides the classes and interfaces for the Java Card security framework.

See:

Description

Interface Summary	
<i>DESKey</i>	DESKey contains an 8/16/24 byte key for single/2 key triple DES/3 key triple DES operations.
<i>DSAKey</i>	The DSAKey interface is the base interface for the DSA algorithms private and public key implementations.
<i>DSAPrivateKey</i>	The DSAPrivateKey interface is used to sign data using the DSA algorithm.
<i>DSAPublicKey</i>	The DSAPublicKey interface is used to verify signatures on signed data using the DSA algorithm.
<i>Key</i>	The Key interface is the base interface for all keys.
<i>PrivateKey</i>	The PrivateKey class is the base class for private keys used in asymmetric algorithms.
<i>PublicKey</i>	The PublicKey class is the base class for public keys used in asymmetric algorithms.
<i>RSAPrivateCrtKey</i>	The RSAPrivateCrtKey interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.
<i>RSAPrivateKey</i>	The RSAPrivateKey class is used to sign data using the RSA algorithm in its modulus/exponent form.
<i>RSAPublicKey</i>	The RSAPublicKey is used to verify signatures on signed data using the RSA algorithm.
<i>SecretKey</i>	The SecretKey class is the base interface for keys used in symmetric algorithms (e.g. DES).

Class Summary	
<i>KeyBuilder</i>	The KeyBuilder class is a key object factory.
<i>MessageDigest</i>	The MessageDigest class is the base class for hashing algorithms.
<i>RandomData</i>	The RandomData abstract class is the base class for random number generation.
<i>Signature</i>	The Signature class is the base class for Signature algorithms.

Exception Summary

CryptoException

CryptoException represents a cryptography-related exception.

Package javacard.security Description

Provides the classes and interfaces for the Java Card security framework.

javacard.security

Class CryptoException

```
java.lang.Object
|
+--java.lang.Throwable
|
+--java.lang.Exception
|
+--java.lang.RuntimeException
|
+--javacard.framework.CardRuntimeException
|
+--javacard.security.CryptoException
```

```
public class CryptoException
extends CardRuntimeException
```

CryptoException represents a cryptography-related exception.

The API classes throw JCRE owned instances of **SystemException**.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components.

See Also:

KeyBuilder, MessageDigest, Signature, RandomData, Cipher

Field Summary

static short	ILLEGAL_USE This reason code is used to indicate that the signature or cipher algorithm does not pad the incoming message and the input message is not block aligned.
static short	ILLEGAL_VALUE This reason code is used to indicate that one or more input parameters is out of allowed bounds.
static short	INVALID_INIT This reason code is used to indicate that the signature or cipher object has not been correctly initialized for the requested operation.
static short	NO_SUCH_ALGORITHM This reason code is used to indicate that the requested algorithm or key type is not supported.
static short	UNINITIALIZED_KEY This reason code is used to indicate that the key is uninitialized.

Constructor Summary

CryptoException(short reason)
 Constructs a CryptoException with the specified reason.

Method Summary

static void **throwIt**(short reason)
 Throws the JCRE owned instance of CryptoException with the specified reason.

Methods inherited from class javacard.framework.CardRuntimeException

getReason, setReason

Methods inherited from class java.lang.Object

equals

Field Detail	
---------------------	--

ILLEGAL_VALUE

```
public static final short ILLEGAL_VALUE
```

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

UNINITIALIZED_KEY

```
public static final short UNINITIALIZED_KEY
```

This reason code is used to indicate that the key is uninitialized.

NO_SUCH_ALGORITHM

```
public static final short NO_SUCH_ALGORITHM
```

This reason code is used to indicate that the requested algorithm or key type is not supported.

INVALID_INIT

```
public static final short INVALID_INIT
```

This reason code is used to indicate that the signature or cipher object has not been correctly initialized for the requested operation.

ILLEGAL_USE

```
public static final short ILLEGAL_USE
```

This reason code is used to indicate that the signature or cipher algorithm does not pad the incoming message and the input message is not block aligned.

Constructor Detail

CryptoException

`public CryptoException(short reason)`

Constructs a `CryptoException` with the specified reason. To conserve on resources use `throwIt()` to use the JCRE owned instance of this class.

Parameters:

`reason` - the reason for the exception.

Method Detail

throwIt

`public static void throwIt(short reason)`

Throws the JCRE owned instance of `CryptoException` with the specified reason.

JCRE owned instances of exception classes are temporary JCRE Entry Point Objects and can be accessed from any applet context. References to these temporary objects cannot be stored in class variables or instance variables or array components. See *Java Card Runtime Environment (JCRE) 2.1 Specification* for details.

Parameters:

`reason` - the reason for the exception.

Throws:

`CryptoException` - always.

javacard.security

Interface DESKey

public abstract interface **DESKey**
 extends **SecretKey**

DESKey contains an 8/16/24 byte key for single/2 key triple DES/3 key triple DES operations.

When the key data is set, the key is initialized and ready for use.

See Also:

KeyBuilder, Signature, Cipher, KeyEncryption

Method Summary

byte	getKey (byte[] keyData, short kOff) Returns the Key data in plain text.
void	setKey (byte[] keyData, short kOff) Sets the Key data.

Methods inherited from interface javacard.security.Key

clearKey, getSize, getType, isInitialized

Method Detail

setKey

```
public void setKey(byte[] keyData,
                   short kOff)
    throws CryptoException
```

Sets the Key data. The plaintext length of input key data is 8 bytes for DES, 16 bytes for 2 key triple DES and 24 bytes for 3 key triple DES. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Parameters:

keyData - byte array containing key initialization data
 kOff - offset within keyData to start

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, keyData is decrypted using the Cipher object.*

getKey

```
public byte getKey(byte[] keyData,  
                  short kOff)
```

Returns the Key data in plain text. The length of output key data is 8 bytes for DES, 16 bytes for 2 key triple DES and 24 bytes for 3 key triple DES. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

keyData - byte array to return key data
kOff - offset within keyData to start.

Returns:

the byte length of the key data returned.

javacard.security

Interface DSAKey

All Known Subinterfaces:

DSAPrivateKey, DSAPublicKey

public abstract interface **DSAKey**

The DSAKey interface is the base interface for the DSA algorithms private and public key implementaions. A DSA private key implementation must also implement the DSAPrivateKey interface methods. A DSA public key implementation must also implement the DSAPublicKey interface methods.

When all four components of the key (X or Y,P,Q,G) are set, the key is initialized and ready for use.

See Also:

DSAPublicKey, DSAPrivateKey, KeyBuilder, Signature, KeyEncryption

Method Summary

short	getG (byte[] buffer, short offset) Returns the subprime parameter value of the key in plain text.
short	getP (byte[] buffer, short offset) Returns the base parameter value of the key in plain text.
short	getQ (byte[] buffer, short offset) Returns the prime parameter value of the key in plain text.
void	setG (byte[] buffer, short offset, short length) Sets the subprime parameter value of the key.
void	setP (byte[] buffer, short offset, short length) Sets the base parameter value of the key.
void	setQ (byte[] buffer, short offset, short length) Sets the prime parameter value of the key.

Method Detail

setP

```
public void setP(byte[] buffer,  
                short offset,  
                short length)  
    throws CryptoException
```

Sets the base parameter value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input base parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the base parameter value begins
length - the length of the base parameter value

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the base parameter value is decrypted using the Cipher object.*
-

setQ

```
public void setQ(byte[] buffer,  
                short offset,  
                short length)  
    throws CryptoException
```

Sets the prime parameter value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input prime parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the prime parameter value begins
length - the length of the prime parameter value

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the prime parameter value is decrypted using the Cipher object.

setG

```
public void setG(byte[] buffer,  
                short offset,  
                short length)  
    throws CryptoException
```

Sets the subprime parameter value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input subprime parameter data is copied into the internal representation.

Parameters:

- buffer - the input buffer
- offset - the offset into the input buffer at which the subprime parameter value begins
- length - the length of the subprime parameter value

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the subprime parameter value is decrypted using the Cipher object.*
-

getP

```
public short getP(byte[] buffer,  
                 short offset)
```

Returns the base parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

- buffer - the output buffer
- offset - the offset into the output buffer at which the base parameter value starts

Returns:

the byte length of the base parameter value returned

getQ

```
public short getQ(byte[] buffer,  
                 short offset)
```

Returns the prime parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the output buffer at which the prime parameter value begins

Returns:

the byte length of the prime parameter value returned

getG

```
public short getG(byte[] buffer,  
                  short offset)
```

Returns the subprime parameter value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the output buffer at which the subprime parameter value begins

Returns:

the byte length of the subprime parameter value returned

javacard.security

Interface DSAPrivateKey

public abstract interface **DSAPrivateKey**
 extends PrivateKey, DSAKey

The DSAPrivateKey interface is used to sign data using the DSA algorithm. An implementation of DSAPrivateKey interface must also implement the DSAKey interface methods.

When all four components of the key (X,P,Q,G) are set, the key is initialized and ready for use.

See Also:

DSAPublicKey, KeyBuilder, Signature, KeyEncryption

Method Summary

short	getX (byte[] buffer, short offset) Returns the value of the key in plain text.
void	setX (byte[] buffer, short offset, short length) Sets the value of the key.

Methods inherited from interface javacard.security.DSAKey

getG, getP, getQ, setG, setP, setQ

Methods inherited from interface javacard.security.Key

clearKey, getSize, getType, isInitialized

Method Detail

setX

```
public void setX(byte[] buffer,  
                short offset,  
                short length)  
    throws CryptoException
```

Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the modulus value begins
length - the length of the modulus

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*
-

getX

```
public short getX(byte[] buffer,  
                 short offset)
```

Returns the value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer
offset - the offset into the output buffer at which the key value starts

Returns:

the byte length of the key value returned

javacard.security

Interface DSAPublicKey

public abstract interface **DSAPublicKey**
 extends **PublicKey**, **DSAKey**

The **DSAPublicKey** interface is used to verify signatures on signed data using the DSA algorithm. An implementation of **DSAPublicKey** interface must also implement the **DSAKey** interface methods.

When all four components of the key (Y,P,Q,G) are set, the key is initialized and ready for use.

See Also:

DSAPrivateKey, **KeyBuilder**, **Signature**, **KeyEncryption**

Method Summary

short	getY (byte[] buffer, short offset) Returns the value of the key in plain text.
void	setY (byte[] buffer, short offset, short length) Sets the value of the key.

Methods inherited from interface javacard.security.DSAKey

getG, getP, getQ, setG, setP, setQ

Methods inherited from interface javacard.security.Key

clearKey, getSize, getType, isInitialized

Method Detail

setY

```
public void setY(byte[] buffer,
                short offset,
                short length)
    throws CryptoException
```

Sets the value of the key. When the base, prime and subprime parameters are initialized and the key value is set, the key is ready for use. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input key data is copied into the internal representation.

Parameters:

buffer - the input buffer
 offset - the offset into the input buffer at which the key value begins
 length - the length of the key value

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input key data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the key value is decrypted using the Cipher object.*

getY

```
public short getY(byte[] buffer,
                 short offset)
```

Returns the value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer
 offset - the offset into the input buffer at which the key value starts

Returns:

the byte length of the key value returned

javacard.security

Interface Key

All Known Subinterfaces:

DESKey, DSAPrivateKey, DSAPublicKey, PrivateKey, PublicKey, RSAPrivateCrtKey, RSAPrivateKey, RSAPublicKey, SecretKey

public abstract interface **Key**

The Key interface is the base interface for all keys.

See Also:

KeyBuilder

Method Summary

void	clearKey() Clears the key and sets its initialized state to false.
short	getSize() Returns the key size in number of bits.
byte	getType() Returns the key interface type.
boolean	isInitialized() Reports the initialized state of the key.

Method Detail

isInitialized

public boolean **isInitialized()**

Reports the initialized state of the key. Keys must be initialized before being used.

A Key object sets its initialized state to true only when all the associated set methods have been invoked at least once since the time the initialized state was set to false.

A newly created Key object sets its initialized state to false. Invocation of the `clearKey()` method sets the initialized state to false. A key with transient key data sets its initialized state to false on the associated clear events.

Returns:

true if the key has been initialized.

clearKey

public void **clearKey**()

Clears the key and sets its initialized state to false.

getType

public byte **getType**()

Returns the key interface type.

Returns:

the key interface type.

See Also:

KeyBuilder

getSize

public short **getSize**()

Returns the key size in number of bits.

Returns:

the key size in number of bits.

javacard.security

Class KeyBuilder

java.lang.Object

+---javacard.security.KeyBuilder

public class **KeyBuilder**

extends Object

The KeyBuilder class is a key object factory.

Field Summary	
static short	LENGTH_DES DES Key Length LENGTH_DES = 64.
static short	LENGTH_DES3_2KEY DES Key Length LENGTH_DES3_2KEY = 128.
static short	LENGTH_DES3_3KEY DES Key Length LENGTH_DES3_3KEY = 192.
static short	LENGTH_DSA_1024 DSA Key Length LENGTH_DSA_1024 = 1024.
static short	LENGTH_DSA_512 DSA Key Length LENGTH_DSA_512 = 512.
static short	LENGTH_DSA_768 DSA Key Length LENGTH_DSA_768 = 768.
static short	LENGTH_RSA_1024 RSA Key Length LENGTH_RSA_1024 = 1024.
static short	LENGTH_RSA_2048 RSA Key Length LENGTH_RSA_2048 = 2048.
static short	LENGTH_RSA_512 RSA Key Length LENGTH_RSA_512 = 512.
static short	LENGTH_RSA_768 RSA Key Length LENGTH_RSA_768 = 768.
static byte	TYPE_DES Key object which implements interface type DESKey with persistent key data.

static byte	TYPE_DES_TRANSIENT_DESELECT Key object which implements interface type DESKey with CLEAR_ON_DESELECT transient key data.
static byte	TYPE_DES_TRANSIENT_RESET Key object which implements interface type DESKey with CLEAR_ON_RESET transient key data.
static byte	TYPE_DSA_PRIVATE Key object which implements the interface type DSAPrivateKey for the DSA algorithm.
static byte	TYPE_DSA_PUBLIC Key object which implements the interface type DSAPublicKey for the DSA algorithm.
static byte	TYPE_RSA_CRT_PRIVATE Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem.
static byte	TYPE_RSA_PRIVATE Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.
static byte	TYPE_RSA_PUBLIC Key object which implements interface type RSAPublicKey.

Method Summary

static Key	buildKey (byte keyType, short keyLength, boolean keyEncryption) Creates cryptographic keys for signature and cipher algorithms.
------------	---

Methods inherited from class java.lang.Object

equals

Field Detail

TYPE_DES_TRANSIENT_RESET

`public static final byte TYPE_DES_TRANSIENT_RESET`

Key object which implements interface type `DESKey` with `CLEAR_ON_RESET` transient key data.

This Key object implicitly performs a `clearKey()` on power on or card reset.

TYPE_DES_TRANSIENT_DESELECT

`public static final byte TYPE_DES_TRANSIENT_DESELECT`

Key object which implements interface type `DESKey` with `CLEAR_ON_DESELECT` transient key data.

This Key object implicitly performs a `clearKey()` on power on, card reset and applet deselection.

TYPE_DES

`public static final byte TYPE_DES`

Key object which implements interface type `DESKey` with persistent key data.

TYPE_RSA_PUBLIC

`public static final byte TYPE_RSA_PUBLIC`

Key object which implements interface type `RSAPublicKey`.

TYPE_RSA_PRIVATE

`public static final byte TYPE_RSA_PRIVATE`

Key object which implements interface type `RSAPrivateKey` which uses modulus/exponent form.

TYPE_RSA_CRT_PRIVATE

`public static final byte TYPE_RSA_CRT_PRIVATE`

Key object which implements interface type `RSAPrivateCrtKey` which uses Chinese Remainder Theorem.

TYPE_DSA_PUBLIC

public static final byte **TYPE_DSA_PUBLIC**

Key object which implements the interface type DSAPublicKey for the DSA algorithm.

TYPE_DSA_PRIVATE

public static final byte **TYPE_DSA_PRIVATE**

Key object which implements the interface type DSAPrivateKey for the DSA algorithm.

LENGTH_DES

public static final short **LENGTH_DES**

DES Key Length **LENGTH_DES** = 64.

LENGTH_DES3_2KEY

public static final short **LENGTH_DES3_2KEY**

DES Key Length **LENGTH_DES3_2KEY** = 128.

LENGTH_DES3_3KEY

public static final short **LENGTH_DES3_3KEY**

DES Key Length **LENGTH_DES3_3KEY** = 192.

LENGTH_RSA_512

public static final short **LENGTH_RSA_512**

RSA Key Length **LENGTH_RSA_512** = 512.

LENGTH_RSA_768

public static final short **LENGTH_RSA_768**

RSA Key Length **LENGTH_RSA_768** = 768.

LENGTH_RSA_1024

public static final short **LENGTH_RSA_1024**

RSA Key Length **LENGTH_RSA_1024** = 1024.

LENGTH_RSA_2048

public static final short **LENGTH_RSA_2048**

RSA Key Length **LENGTH_RSA_2048** = 2048.

LENGTH_DSA_512

public static final short **LENGTH_DSA_512**

DSA Key Length **LENGTH_DSA_512** = 512.

LENGTH_DSA_768

public static final short **LENGTH_DSA_768**

DSA Key Length **LENGTH_DSA_768** = 768.

LENGTH_DSA_1024

public static final short **LENGTH_DSA_1024**

DSA Key Length **LENGTH_DSA_1024** = 1024.

<h2>Method Detail</h2>

buildKey

```
public static Key buildKey(byte keyType,  
                           short keyLength,  
                           boolean keyEncryption)  
    throws CryptoException
```

Creates cryptographic keys for signature and cipher algorithms. Instances created by this method may be the only key objects used to initialize instances of `Signature` and `Cipher`. Note that the object returned must be cast to their appropriate key type interface.

Parameters:

`keyType` - the type of key to be generated. Valid codes listed in `TYPE..` constants.

`keyLength` - the key size in bits. The valid key bit lengths are key type dependent. See above.

`keyEncryption` - if true this boolean requests a key implementation which implements the `javacardx.cipher.KeyEncryption` interface.

Returns:

the key object instance of the requested key type, length and encrypted access.

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm associated with the specified type, size of key and key encryption interface is not supported.
-
-

javacard.security

Class MessageDigest

java.lang.Object

|
+---javacard.security.MessageDigest

public abstract class MessageDigest
extends Object

The MessageDigest class is the base class for hashing algorithms. Implementations of MessageDigest algorithms must extend this class and implement all the abstract methods.

Field Summary

static byte	ALG_MD5 Message Digest algorithm MD5.
static byte	ALG_RIPEMD160 Message Digest algorithm RIPE MD-160.
static byte	ALG_SHA Message Digest algorithm SHA.

Constructor Summary

protected	MessageDigest () Protected Constructor
-----------	--

Method Summary

abstract short	doFinal (byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) Generates a hash of all/last input data.
abstract byte	getAlgorithm () Gets the Message digest algorithm.
static MessageDigest	getInstance (byte algorithm, boolean externalAccess) Creates a MessageDigest object instance of the selected algorithm.
abstract byte	getLength () Returns the byte length of the hash.
abstract void	reset () Resets the MessageDigest object to the initial state for further use.
abstract void	update (byte[] inBuff, short inOffset, short inLength) Accumulates a hash of the input data.

Methods inherited from class java.lang.Object

equals

Field Detail

ALG_SHA

public static final byte **ALG_SHA**

Message Digest algorithm SHA.

ALG_MD5

public static final byte **ALG_MD5**

Message Digest algorithm MD5.

ALG_RIPEMD160

public static final byte **ALG_RIPEMD160**

Message Digest algorithm RIPE MD-160.

Constructor Detail

MessageDigest

protected **MessageDigest**()

Protected Constructor

Method Detail

getInstance

public static final MessageDigest **getInstance**(byte algorithm,
boolean externalAccess)
throws **CryptoException**

Creates a MessageDigest object instance of the selected algorithm.

Parameters:

algorithm - the desired message digest algorithm. Valid codes listed in ALG_.. constants. See above.

externalAccess - if true indicates that the instance will be shared among multiple applet instances and that the MessageDigest instance will also be accessed (via a Shareable interface) when the owner of the MessageDigest instance is not the currently selected applet.

Returns:

the MessageDigest object instance of the requested algorithm.

Throws:

CryptoException - with the following reason codes:

- **CryptoException.NO_SUCH_ALGORITHM** if the requested algorithm is not supported.

getAlgorithm

public abstract byte **getAlgorithm**()

Gets the Message digest algorithm.

Returns:

the algorithm code defined above.

getLength

```
public abstract byte getLength()
```

Returns the byte length of the hash.

Returns:

hash length

doFinal

```
public abstract short doFinal(byte[] inBuff,  
                              short inOffset,  
                              short inLength,  
                              byte[] outBuff,  
                              short outOffset)
```

Generates a hash of all/last input data. Completes and returns the hash computation after performing final operations such as padding. The `MessageDigest` object is reset after this call is made.

The input and output buffer data may overlap.

Parameters:

`inBuff` - the input buffer of data to be hashed

`inOffset` - the offset into the input buffer at which to begin hash generation

`inLength` - the byte length to hash

`outBuff` - the output buffer, may be the same as the input buffer

`outOffset` - the offset into the output buffer where the resulting hash value begins

Returns:

number of bytes of hash output in `outBuff`

update

```
public abstract void update(byte[] inBuff,  
                             short inOffset,  
                             short inLength)
```

Accumulates a hash of the input data. When this method is used temporary storage of intermediate results is required. This method should only be used if all the input data required for the hash is not available in one byte array. The `doFinal()` method is recommended whenever possible.

Parameters:

`inBuff` - the input buffer of data to be hashed

`inOffset` - the offset into the input buffer at which to begin hash generation

`inLength` - the byte length to hash

See Also:

`doFinal(byte[], short, short, byte[], short)`

reset

```
public abstract void reset()
```

Resets the MessageDigest object to the initial state for further use.

javacard.security

Interface PrivateKey

All Known Subinterfaces:

DSAPrivateKey, RSAPrivateCrtKey, RSAPrivateKey

public abstract interface **PrivateKey**
extends Key

The PrivateKey class is the base class for private keys used in asymmetric algorithms.

Methods inherited from interface javacard.security.Key

clearKey, getSize, getType, isInitialized

javacard.security

Interface PublicKey

All Known Subinterfaces:

DSAPublicKey, RSAPublicKey

public abstract interface **PublicKey**
extends Key

The `PublicKey` class is the base class for public keys used in asymmetric algorithms.

Methods inherited from interface javacard.security.Key

<code>clearKey</code> , <code>getSize</code> , <code>getType</code> , <code>isInitialized</code>
--

javacard.security

Interface RSAPrivateCrtKey

public abstract interface **RSAPrivateCrtKey**
extends **PrivateKey**

The **RSAPrivateCrtKey** interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

Let $S = m^d \bmod n$, where m is the data to be signed, d is the private key exponent, and n is private key modulus composed of two prime numbers p and q . The following names are used in the initializer methods in this interface:

P, the prime factor p

Q, the prime factor q .

$PQ = q^{-1} \bmod p$

$DP1 = d \bmod (p - 1)$

$DQ1 = d \bmod (q - 1)$

When all five components (P,Q,PQ,DP1,DQ1) of the key are set, the key is initialized and ready for use.

See Also:

`RSAPrivateKey`, `RSAPublicKey`, `KeyBuilder`, `Signature`, `Cipher`, `KeyEncryption`

Method Summary

short	getDP1 (byte[] buffer, short offset) Returns the value of the DP1 parameter in plain text.
short	getDQ1 (byte[] buffer, short offset) Returns the value of the DQ1 parameter in plain text.
short	getP (byte[] buffer, short offset) Returns the value of the P parameter in plain text.
short	getPQ (byte[] buffer, short offset) Returns the value of the PQ parameter in plain text.
short	getQ (byte[] buffer, short offset) Returns the value of the Q parameter in plain text.
void	setDP1 (byte[] buffer, short offset, short length) Sets the value of the DP1 parameter.
void	setDQ1 (byte[] buffer, short offset, short length) Sets the value of the DQ1 parameter.
void	setP (byte[] buffer, short offset, short length) Sets the value of the P parameter.
void	setPQ (byte[] buffer, short offset, short length) Sets the value of the PQ parameter.
void	setQ (byte[] buffer, short offset, short length) Sets the value of the Q parameter.

Methods inherited from interface javacard.security.Key

clearKey, getSize, getType, isInitialized

Method Detail

setP

```
public void setP(byte[] buffer,
                short offset,
                short length)
    throws CryptoException
```

Sets the value of the P parameter. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input P parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the parameter value begins
length - the length of the parameter

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the P parameter value is decrypted using the Cipher object.*
-

setQ

```
public void setQ(byte[] buffer,  
                short offset,  
                short length)  
    throws CryptoException
```

Sets the value of the Q parameter. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input Q parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the parameter value begins
length - the length of the parameter

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the Q parameter value is decrypted using the Cipher object.*
-

setDP1

```
public void setDP1(byte[] buffer,
                  short offset,
                  short length)
    throws CryptoException
```

Sets the value of the DP1 parameter. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DP1 parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
 offset - the offset into the input buffer at which the parameter value begins
 length - the length of the parameter

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the DP1 parameter value is decrypted using the Cipher object.*

setDQ1

```
public void setDQ1(byte[] buffer,
                  short offset,
                  short length)
    throws CryptoException
```

Sets the value of the DQ1 parameter. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input DQ1 parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
 offset - the offset into the input buffer at which the parameter value begins
 length - the length of the parameter

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the DQ1 parameter value is decrypted using the Cipher object.*

setPQ

```
public void setPQ(byte[] buffer,
                 short offset,
                 short length)
    throws CryptoException
```

Sets the value of the PQ parameter. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input PQ parameter data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the parameter value begins
length - the length of the parameter

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input parameter data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the PQ parameter value is decrypted using the Cipher object.*
-

getP

```
public short getP(byte[] buffer,
                 short offset)
```

Returns the value of the P parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer
offset - the offset into the output buffer at which the parameter value begins

Returns:

the byte length of the P parameter value returned

getQ

```
public short getQ(byte[] buffer,
                 short offset)
```

Returns the value of the Q parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

Returns:

the byte length of the Q parameter value returned

getDP1

```
public short getDP1(byte[] buffer,
                    short offset)
```

Returns the value of the DP1 parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

Returns:

the byte length of the DP1 parameter value returned

getDQ1

```
public short getDQ1(byte[] buffer,
                    short offset)
```

Returns the value of the DQ1 parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

Returns:

the byte length of the DQ1 parameter value returned

getPQ

```
public short getPQ(byte[] buffer,
                   short offset)
```

Returns the value of the PQ parameter in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

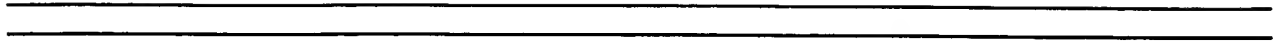
Parameters:

`buffer` - the output buffer

`offset` - the offset into the output buffer at which the parameter value begins

Returns:

the byte length of the PQ parameter value returned



javacard.security**Interface RSAPrivateKey**

public abstract interface **RSAPrivateKey**
 extends **PrivateKey**

The **RSAPrivateKey** class is used to sign data using the RSA algorithm in its modulus/exponent form. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

When both the modulus and exponent of the key are set, the key is initialized and ready for use.

See Also:

RSAPublicKey, **RSAPrivateCrtKey**, **KeyBuilder**, **Signature**, **Cipher**,
KeyEncryption

Method Summary

short	getExponent (byte[] buffer, short offset) Returns the private exponent value of the key in plain text.
short	getModulus (byte[] buffer, short offset) Returns the modulus value of the key in plain text.
void	setExponent (byte[] buffer, short offset, short length) Sets the private exponent value of the key.
void	setModulus (byte[] buffer, short offset, short length) Sets the modulus value of the key.

Methods inherited from interface javacard.security.Key

clearKey, **getSize**, **getType**, **isInitialized**

Method Detail

setModulus

```
public void setModulus(byte[] buffer,  
                      short offset,  
                      short length)  
    throws CryptoException
```

Sets the modulus value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input modulus data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the modulus value begins
length - the length of the modulus

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input modulus data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the modulus value is decrypted using the Cipher object.*
-

setExponent

```
public void setExponent(byte[] buffer,  
                       short offset,  
                       short length)  
    throws CryptoException
```

Sets the private exponent value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input exponent data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the exponent value begins
length - the length of the exponent

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input exponent data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the exponent value is decrypted using the Cipher object.*

getModulus

```
public short getModulus(byte[] buffer,  
                        short offset)
```

Returns the modulus value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the output buffer at which the modulus value starts

Returns:

the byte length of the modulus value returned

getExponent

```
public short getExponent(byte[] buffer,  
                        short offset)
```

Returns the private exponent value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the output buffer at which the exponent value begins

Returns:

the byte length of the private exponent value returned

javacard.security

Interface RSAPublicKey

public abstract interface **RSAPublicKey**
extends **PublicKey**

The **RSAPublicKey** is used to verify signatures on signed data using the RSA algorithm. It may also be used by the `javacardx.crypto.Cipher` class to encrypt/decrypt messages.

When both the modulus and exponent of the key are set, the key is initialized and ready for use.

See Also:

`RSAPrivateKey`, `RSAPrivateCrtKey`, `KeyBuilder`, `Signature`, `Cipher`,
`KeyEncryption`

Method Summary

short	getExponent (byte[] buffer, short offset) Returns the private exponent value of the key in plain text.
short	getModulus (byte[] buffer, short offset) Returns the modulus value of the key in plain text.
void	setExponent (byte[] buffer, short offset, short length) Sets the public exponent value of the key.
void	setModulus (byte[] buffer, short offset, short length) Sets the modulus value of the key.

Methods inherited from interface javacard.security.Key

`clearKey`, `getSize`, `getType`, `isInitialized`

Method Detail

setModulus

```
public void setModulus(byte[] buffer,
                      short offset,
                      short length)
    throws CryptoException
```

Sets the modulus value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input modulus data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the modulus value begins
length - the byte length of the modulus

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input modulus data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

- *If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the modulus value is decrypted using the Cipher object.*

setExponent

```
public void setExponent(byte[] buffer,
                       short offset,
                       short length)
    throws CryptoException
```

Sets the public exponent value of the key. The plaintext data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte). Input exponent data is copied into the internal representation.

Parameters:

buffer - the input buffer
offset - the offset into the input buffer at which the exponent value begins
length - the byte length of the exponent

Throws:

CryptoException - with the following reason code:

- `CryptoException.ILLEGAL_VALUE` if the input exponent data length is inconsistent with the implementation or if input data decryption is required and fails.

Note:

If the key object implements the `javacardx.crypto.KeyEncryption` interface and the Cipher object specified via `setKeyCipher()` is not null, the exponent value is decrypted using the Cipher object.

getModulus

```
public short getModulus(byte[] buffer,  
                        short offset)
```

Returns the modulus value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the input buffer at which the modulus value starts

Returns:

the byte length of the modulus value returned

getExponent

```
public short getExponent(byte[] buffer,  
                        short offset)
```

Returns the private exponent value of the key in plain text. The data format is big-endian and right-aligned (the least significant bit is the least significant bit of last byte).

Parameters:

buffer - the output buffer

offset - the offset into the output buffer at which the exponent value begins

Returns:

the byte length of the public exponent returned

javacard.security

Class RandomData

java.lang.Object

|--javacard.security.RandomData

public abstract class **RandomData**

extends Object

The RandomData abstract class is the base class for random number generation. Implementations of RandomData algorithms must extend this class and implement all the abstract methods.

Field Summary

static byte	ALG_PSEUDO_RANDOM Utility pseudo random number generation algorithms.
static byte	ALG_SECURE_RANDOM Cryptographically secure random number generation algorithms.

Constructor Summary

protected	RandomData () Protected constructor for subclassing.
-----------	--

Method Summary

abstract void	generateData (byte[] buffer, short offset, short length) Generates random data.
static RandomData	getInstance (byte algorithm) Creates a RandomData instance of the selected algorithm.
abstract void	setSeed (byte[] buffer, short offset, short length) Seeds the random data generator.

Methods inherited from class java.lang.Object

`equals`

Field Detail

ALG_PSEUDO_RANDOM

public static final byte **ALG_PSEUDO_RANDOM**

Utility pseudo random number generation algorithms.

ALG_SECURE_RANDOM

public static final byte **ALG_SECURE_RANDOM**

Cryptographically secure random number generation algorithms.

Constructor Detail

RandomData

protected **RandomData**()

Protected constructor for subclassing.

Method Detail

getInstance

public static final RandomData **getInstance**(byte algorithm)
throws **CryptoException**

Creates a RandomData instance of the selected algorithm. The pseudo random RandomData instance's seed is initialized to a internal default value.

Parameters:

algorithm - the desired random number algorithm. Valid codes listed in ALG_.. constants. See above.

Returns:

the RandomData object instance of the requested algorithm.

Throws:

CryptoException - with the following reason codes:

`CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported.

generateData

```
public abstract void generateData(byte[] buffer,  
                                   short offset,  
                                   short length)
```

Generates random data.

Parameters:

`buffer` - the output buffer
`offset` - the offset into the output buffer
`length` - the length of random data to generate

setSeed

```
public abstract void setSeed(byte[] buffer,  
                              short offset,  
                              short length)
```

Seeds the random data generator.

Parameters:

`buffer` - the input buffer
`offset` - the offset into the input buffer
`length` - the length of the seed data

javacard.security

Interface SecretKey

All Known Subinterfaces:
DESKey

public abstract interface **SecretKey**
extends **Key**

The **SecretKey** class is the base interface for keys used in symmetric algorithms (e.g. DES).

Methods inherited from interface javacard.security.Key

<code>clearKey, getSize, getType, isInitialized</code>
--

javacard.security

Class Signature

java.lang.Object

---javacard.security.Signature

public abstract class **Signature**
extends Object

The **Signature** class is the base class for Signature algorithms. Implementations of **Signature** algorithms must extend this class and implement all the abstract methods.

The term "pad" is used in the public key signature algorithms below to refer to all the operations specified in the referenced scheme to transform the message digest into the encryption block size.

Field Summary	
static byte	ALG_DES_MAC4_ISO9797_M1 Signature algorithm ALG_DES_MAC4_ISO9797_M1 generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.
static byte	ALG_DES_MAC4_ISO9797_M2 Signature algorithm ALG_DES_MAC4_ISO9797_M2 generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
static byte	ALG_DES_MAC4_NOPAD Signature algorithm ALG_DES_MAC4_NOPAD generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.
static byte	ALG_DES_MAC4_PKCS5 Signature algorithm ALG_DES_MAC4_PKCS5 generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.
static byte	ALG_DES_MAC8_ISO9797_M1 Signature algorithm ALG_DES_MAC8_ISO9797_M1 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

static byte	ALG_DES_MAC8_ISO9797_M2 Signature algorithm ALG_DES_MAC8_ISO9797_M2 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
static byte	ALG_DES_MAC8_NOPAD Signature algorithm ALG_DES_MAC_8_NOPAD generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.
static byte	ALG_DES_MAC8_PKCS5 Signature algorithm ALG_DES_MAC8_PKCS5 generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.
static byte	ALG_DSA_SHA Signature algorithm ALG_DSA_SHA signs/verifies the 20 byte SHA digest using DSA.
static byte	ALG_RSA_MD5_PKCS1 Signature algorithm ALG_RSA_MD5_PKCS1 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.
static byte	ALG_RSA_MD5_RFC2409 Signature algorithm ALG_RSA_MD5_RFC2409 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the RFC2409 scheme.
static byte	ALG_RSA_RIPEMD160_ISO9796 Signature algorithm ALG_RSA_RIPEMD160_ISO9796 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the ISO 9796 scheme.
static byte	ALG_RSA_RIPEMD160_PKCS1 Signature algorithm ALG_RSA_RIPEMD160_PKCS1 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.
static byte	ALG_RSA_SHA_ISO9796 Signature algorithm ALG_RSA_SHA_ISO9796 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the ISO 9796 (EMV'96) scheme.
static byte	ALG_RSA_SHA_PKCS1 Signature algorithm ALG_RSA_SHA_PKCS1 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.
static byte	ALG_RSA_SHA_RFC2409 Signature algorithm ALG_RSA_SHA_RFC2409 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the RFC2409 scheme.
static byte	MODE_SIGN Used in init() methods to indicate signature sign mode.

static byte	MODE_VERIFY Used in <code>init()</code> methods to indicate signature verify mode.
-------------	--

Constructor Summary

protected	Signature() Protected Constructor
-----------	---

Method Summary

abstract byte	getAlgorithm() Gets the Signature algorithm.
static Signature	getInstance (byte algorithm, boolean externalAccess) Creates a Signature object instance of the selected algorithm.
abstract short	getLength() Returns the byte length of the signature data.
abstract void	init (Key theKey, byte theMode) Initializes the Signature object with the appropriate Key.
abstract void	init (Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) Initializes the Signature object with the appropriate Key and algorithm specific parameters.
abstract short	sign (byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset) Generates the signature of all/last input data.
abstract void	update (byte[] inBuff, short inOffset, short inLength) Accumulates a signature of the input data.
abstract boolean	verify (byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength) Verifies the signature of all/last input data against the passed in signature.

Methods inherited from class java.lang.Object

`equals`

Field Detail	
---------------------	--

ALG_DES_MAC4_NOPAD

```
public static final byte ALG_DES_MAC4_NOPAD
```

Signature algorithm `ALG_DES_MAC4_NOPAD` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_MAC8_NOPAD

```
public static final byte ALG_DES_MAC8_NOPAD
```

Signature algorithm `ALG_DES_MAC_8_NOPAD` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

Note:

- *This algorithm must not be implemented if export restrictions apply.*
-

ALG_DES_MAC4_ISO9797_M1

```
public static final byte ALG_DES_MAC4_ISO9797_M1
```

Signature algorithm `ALG_DES_MAC4_ISO9797_M1` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_MAC8_ISO9797_M1

```
public static final byte ALG_DES_MAC8_ISO9797_M1
```

Signature algorithm `ALG_DES_MAC8_ISO9797_M1` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*

ALG_DES_MAC4_ISO9797_M2

public static final byte **ALG_DES_MAC4_ISO9797_M2**

Signature algorithm **ALG_DES_MAC4_ISO9797_M2** generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_MAC8_ISO9797_M2

public static final byte **ALG_DES_MAC8_ISO9797_M2**

Signature algorithm **ALG_DES_MAC8_ISO9797_M2** generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*
-

ALG_DES_MAC4_PKCS5

public static final byte **ALG_DES_MAC4_PKCS5**

Signature algorithm **ALG_DES_MAC4_PKCS5** generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

ALG_DES_MAC8_PKCS5

public static final byte **ALG_DES_MAC8_PKCS5**

Signature algorithm **ALG_DES_MAC8_PKCS5** generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

Note:

- *This algorithm must not be implemented if export restrictions apply.*
-

ALG_RSA_SHA_ISO9796

public static final byte **ALG_RSA_SHA_ISO9796**

Signature algorithm ALG_RSA_SHA_ISO9796 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the ISO 9796 (EMV'96) scheme.

ALG_RSA_SHA_PKCS1

public static final byte **ALG_RSA_SHA_PKCS1**

Signature algorithm ALG_RSA_SHA_PKCS1 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_RSA_MD5_PKCS1

public static final byte **ALG_RSA_MD5_PKCS1**

Signature algorithm ALG_RSA_MD5_PKCS1 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_RSA_RIPEMD160_ISO9796

public static final byte **ALG_RSA_RIPEMD160_ISO9796**

Signature algorithm ALG_RSA_RIPEMD160_ISO9796 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the ISO 9796 scheme.

ALG_RSA_RIPEMD160_PKCS1

public static final byte **ALG_RSA_RIPEMD160_PKCS1**

Signature algorithm ALG_RSA_RIPEMD160_PKCS1 encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_DSA_SHA

public static final byte **ALG_DSA_SHA**

Signature algorithm ALG_DSA_SHA signs/verifies the 20 byte SHA digest using DSA.

ALG_RSA_SHA_RFC2409

public static final byte **ALG_RSA_SHA_RFC2409**

Signature algorithm ALG_RSA_SHA_RFC2409 encrypts the 20 byte SHA digest using RSA. The digest is padded according to the RFC2409 scheme.

ALG_RSA_MD5_RFC2409

public static final byte **ALG_RSA_MD5_RFC2409**

Signature algorithm ALG_RSA_MD5_RFC2409 encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the RFC2409 scheme.

MODE_SIGN

public static final byte **MODE_SIGN**

Used in `init()` methods to indicate signature sign mode.

MODE_VERIFY

public static final byte **MODE_VERIFY**

Used in `init()` methods to indicate signature verify mode.

Constructor Detail

Signature

protected **Signature()**

Protected Constructor

Method Detail

getInstance

public static final Signature **getInstance**(byte algorithm,
boolean externalAccess)
throws **CryptoException**

Creates a Signature object instance of the selected algorithm.

Parameters:

- algorithm - the desired Signature algorithm. See above.
- externalAccess - if true indicates that the instance will be shared among multiple applet instances and that the Signature instance will also be accessed (via a Shareable interface) when the owner of the Signature instance is not the currently selected applet.

Returns:

the Signature object instance of the requested algorithm.

Throws:

CryptoException - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported.

init

```
public abstract void init(Key theKey,  
                          byte theMode)  
    throws CryptoException
```

Initializes the Signature object with the appropriate Key. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

Note:

- *DES and triple DES algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.*

Parameters:

- theKey - the key object to use for signing or verifying
- theMode - one of `MODE_SIGN` or `MODE_VERIFY`

Throws:

CryptoException - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if theMode option is an undefined value or if the Key is inconsistent with theMode or with the Signature implementation.

init

```
public abstract void init(Key theKey,  
                          byte theMode,  
                          byte[] bArray,  
                          short bOff,  
                          short bLen)  
    throws CryptoException
```

Initializes the Signature object with the appropriate Key and algorithm specific parameters.

Note:

DES and triple DES algorithms in outer CBC mode expect an 8 byte parameter value for the initial vector(IV) in bArray.

RSA and DSA algorithms throw `CryptoException.ILLEGAL_VALUE`.

Parameters:

`theKey` - the key object to use for signing
`theMode` - one of `MODE_SIGN` or `MODE_VERIFY`
`bArray` - byte array containing algorithm specific initialization info.
`bOff` - offset within `bArray` where the algorithm specific data begins.
`bLen` - byte length of algorithm specific parameter data

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if `theMode` option is an undefined value or if a byte array parameter option is not supported by the algorithm or if the `bLen` is an incorrect byte length for the algorithm specific data or if the `Key` is inconsistent with `theMode` or with the Signature implementation.

getAlgorithm

```
public abstract byte getAlgorithm()
```

Gets the Signature algorithm.

Returns:

the algorithm code defined above.

getLength

```
public abstract short getLength()
```

Returns the byte length of the signature data.

Returns:

the byte length of the signature data.

update

```
public abstract void update(byte[] inBuff,
                           short inOffset,
                           short inLength)
    throws CryptoException
```

Accumulates a signature of the input data. When this method is used temporary storage of intermediate results is required. This method should only be used if all the input data required for the signature is not available in one byte array. The `sign()` or `verify()` method is recommended whenever possible.

Parameters:

`inBuff` - the input buffer of data to be signed
`inOffset` - the offset into the input buffer at which to begin signature generation
`inLength` - the byte length to sign

Throws:

CryptoException - with the following reason codes:

- CryptoException.UNINITIALIZED_KEY if key not initialized.

See Also:

sign(byte[], short, short, byte[], short), verify(byte[], short, short, byte[], short, short)

sign

```
public abstract short sign(byte[] inBuff,
                           short inOffset,
                           short inLength,
                           byte[] sigBuff,
                           short sigOffset)
    throws CryptoException
```

Generates the signature of all/last input data. A call to this method also resets this Signature object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to sign another message.

The input and output buffer data may overlap.

Parameters:

inBuff - the input buffer of data to be signed
inOffset - the offset into the input buffer at which to begin signature generation
inLength - the byte length to sign
sigBuff - the output buffer to store signature data
sigOffset - the offset into sigBuff at which to begin signature data

Returns:

number of bytes of signature output in sigBuff

Throws:

CryptoException - with the following reason codes:

- CryptoException.UNINITIALIZED_KEY if key not initialized.
 - CryptoException.INVALID_INIT if this Signature object is not initialized or initialized for signature verify mode.
 - CryptoException.ILLEGAL_USE if this Signature algorithm does not pad the message and the message is not block aligned.
-

verify

```
public abstract boolean verify(byte[] inBuff,
                               short inOffset,
                               short inLength,
                               byte[] sigBuff,
                               short sigOffset,
                               short sigLength)
    throws CryptoException
```

Verifies the signature of all/last input data against the passed in signature. A call to this method also resets this `Signature` object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to verify another message.

Parameters:

`inBuff` - the input buffer of data to be verified
`inOffset` - the offset into the input buffer at which to begin signature generation
`inLength` - the byte length to sign
`sigBuff` - the input buffer containing signature data
`sigOffset` - the offset into `sigBuff` where signature data begins.
`sigLength` - the byte length of the signature data

Returns:

true if signature verifies false otherwise.

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
 - `CryptoException.INVALID_INIT` if this `Signature` object is not initialized or initialized for signature sign mode.
 - `CryptoException.ILLEGAL_USE` if this `Signature` algorithm does not pad the message and the message is not block aligned.
-
-

Package javacardx.crypto

Extension package containing security classes and interfaces for export-controlled functionality.

See:

Description

Interface Summary	
-------------------	--

<i>KeyEncryption</i>	<code>KeyEncryption</code> interface defines the methods used to enable encrypted key data access to a key implementation.
----------------------	--

Class Summary	
---------------	--

<code>Cipher</code>	The <code>Cipher</code> class is the abstract base class for Cipher algorithms.
---------------------	---

Package javacardx.crypto Description

Extension package containing security classes and interfaces for export-controlled functionality.

javacardx.crypto

Class Cipher

java.lang.Object

---javacardx.crypto.Cipher

public abstract class Cipher
extends Object

The Cipher class is the abstract base class for Cipher algorithms. Implementations of Cipher algorithms must extend this class and implement all the abstract methods.

The term "pad" is used in the public key cipher algorithms below to refer to all the operations specified in the referenced scheme to transform the message block into the cipher block size.

Field Summary

static byte	ALG_DES_CBC_ISO9797_M1 Cipher algorithm ALG_DES_CBC_ISO9797_M1 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.
static byte	ALG_DES_CBC_ISO9797_M2 Cipher algorithm ALG_DES_CBC_ISO9797_M2 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
static byte	ALG_DES_CBC_NOPAD Cipher algorithm ALG_DES_CBC_NOPAD provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.
static byte	ALG_DES_CBC_PKCS5 Cipher algorithm ALG_DES_CBC_PKCS5 provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.
static byte	ALG_DES_ECB_ISO9797_M1 Cipher algorithm ALG_DES_ECB_ISO9797_M1 provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 1 scheme.
static byte	ALG_DES_ECB_ISO9797_M2 Cipher algorithm ALG_DES_ECB_ISO9797_M2 provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

static byte	ALG_DES_ECB_NOPAD Cipher algorithm ALG_DES_ECB_NOPAD provides a cipher using DES in ECB mode. This algorithm does not pad input data.
static byte	ALG_DES_ECB_PKCS5 Cipher algorithm ALG_DES_ECB_PKCS5 provides a cipher using DES in ECB mode. Input data is padded according to the PKCS#5 scheme.
static byte	ALG_RSA_ISO14888 Cipher algorithm ALG_RSA_ISO14888 provides a cipher using RSA. Input data is padded according to the ISO 14888 scheme.
static byte	ALG_RSA_ISO9796 Cipher algorithm ALG_RSA_ISO9796 provides a cipher using RSA. Input data is padded according to the ISO 9796 (EMV'96) scheme.
static byte	ALG_RSA_PKCS1 Cipher algorithm ALG_RSA_PKCS1 provides a cipher using RSA. Input data is padded according to the PKCS#1 (v1.5) scheme.
static byte	MODE_DECRYPT Used in <code>init()</code> methods to indicate decryption mode.
static byte	MODE_ENCRYPT Used in <code>init()</code> methods to indicate encryption mode.

Constructor Summary

protected	Cipher() Protected Constructor
-----------	--

Method Summary

abstract short	doFinal (byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) Generates encrypted/decrypted output from all/last input data.
abstract byte	getAlgorithm () Gets the Cipher algorithm.
static Cipher	getInstance (byte algorithm, boolean externalAccess) Creates a Cipher object instance of the selected algorithm.
abstract void	init (Key theKey, byte theMode) Initializes the Cipher object with the appropriate Key.
abstract void	init (Key theKey, byte theMode, byte[] bArray, short bOff, short bLen) Initializes the Cipher object with the appropriate Key and algorithm specific parameters.
abstract short	update (byte[] inBuff, short inOffset, short inLength, byte[] outBuff, short outOffset) Generates encrypted/decrypted output from input data.

Methods inherited from class java.lang.Object

equals

Field Detail

ALG_DES_CBC_NOPAD

```
public static final byte ALG_DES_CBC_NOPAD
```

Cipher algorithm ALG_DES_CBC_NOPAD provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

ALG_DES_CBC_ISO9797_M1

```
public static final byte ALG_DES_CBC_ISO9797_M1
```

Cipher algorithm `ALG_DES_CBC_ISO9797_M1` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

`ALG_DES_CBC_ISO9797_M2`

```
public static final byte ALG_DES_CBC_ISO9797_M2
```

Cipher algorithm `ALG_DES_CBC_ISO9797_M2` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

`ALG_DES_CBC_PKCS5`

```
public static final byte ALG_DES_CBC_PKCS5
```

Cipher algorithm `ALG_DES_CBC_PKCS5` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

`ALG_DES_ECB_NOPAD`

```
public static final byte ALG_DES_ECB_NOPAD
```

Cipher algorithm `ALG_DES_ECB_NOPAD` provides a cipher using DES in ECB mode. This algorithm does not pad input data. If the input data is not (8 byte) block aligned it throws `CryptoException` with the reason code `ILLEGAL_USE`.

`ALG_DES_ECB_ISO9797_M1`

```
public static final byte ALG_DES_ECB_ISO9797_M1
```

Cipher algorithm `ALG_DES_ECB_ISO9797_M1` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 1 scheme.

`ALG_DES_ECB_ISO9797_M2`

```
public static final byte ALG_DES_ECB_ISO9797_M2
```

Cipher algorithm `ALG_DES_ECB_ISO9797_M2` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_ECB_PKCS5

```
public static final byte ALG_DES_ECB_PKCS5
```

Cipher algorithm ALG_DES_ECB_PKCS5 provides a cipher using DES in ECB mode. Input data is padded according to the PKCS#5 scheme.

ALG_RSA_ISO14888

```
public static final byte ALG_RSA_ISO14888
```

Cipher algorithm ALG_RSA_ISO14888 provides a cipher using RSA. Input data is padded according to the ISO 14888 scheme.

ALG_RSA_PKCS1

```
public static final byte ALG_RSA_PKCS1
```

Cipher algorithm ALG_RSA_PKCS1 provides a cipher using RSA. Input data is padded according to the PKCS#1 (v1.5) scheme.

Note:

- *This algorithm is only suitable for messages of limited length. The total number of input bytes processed may not be more than $k-11$, where k is the RSA key's modulus size in bytes.*
-

ALG_RSA_ISO9796

```
public static final byte ALG_RSA_ISO9796
```

Cipher algorithm ALG_RSA_ISO9796 provides a cipher using RSA. Input data is padded according to the ISO 9796 (EMV'96) scheme.

Note:

- *This algorithm is only suitable for messages of limited length. The total number of input bytes processed may not be more than $k/2$, where k is the RSA key's modulus size in bytes.*
-

MODE_DECRYPT

```
public static final byte MODE_DECRYPT
```

Used in `init()` methods to indicate decryption mode.

MODE_ENCRYPT

```
public static final byte MODE_ENCRYPT
```

Used in `init()` methods to indicate encryption mode.

Constructor Detail

Cipher

```
protected Cipher()
```

Protected Constructor

Method Detail

getInstance

```
public static final Cipher getInstance(byte algorithm,  
                                         boolean externalAccess)  
    throws CryptoException
```

Creates a `Cipher` object instance of the selected algorithm.

Parameters:

`algorithm` - the desired `Cipher` algorithm. See above.
`externalAccess` - if true indicates that the instance will be shared among multiple applet instances and that the `Cipher` instance will also be accessed (via a `Shareable` interface) when the owner of the `Cipher` instance is not the currently selected applet.

Returns:

the `Cipher` object instance of the requested algorithm.

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.NO_SUCH_ALGORITHM` if the requested algorithm is not supported.

init

```
public abstract void init(Key theKey,  
                           byte theMode)  
    throws CryptoException
```

Initializes the `Cipher` object with the appropriate `Key`. This method should be used for algorithms which do not need initialization parameters or use default parameter values.

Note:

DES and triple DES algorithms in CBC mode will use 0 for initial vector(IV) if this method is used.

Parameters:

theKey - the key object to use for signing or verifying

theMode - one of MODE_DECRYPT or MODE_ENCRYPT

Throws:

CryptoException - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if theMode option is an undefined value or if the Key is inconsistent with the Cipher implementation.

init

```
public abstract void init(Key theKey,
                        byte theMode,
                        byte[] bArray,
                        short bOff,
                        short bLen)
    throws CryptoException
```

Initializes the Cipher object with the appropriate Key and algorithm specific parameters.

Note:

- *DES and triple DES algorithms in outer CBC mode expect an 8 byte parameter value for the initial vector(IV) in bArray.*
- *RSA and DSA algorithms throw `CryptoException.ILLEGAL_VALUE`.*

Parameters:

theKey - the key object to use for signing

theMode - one of MODE_DECRYPT or MODE_ENCRYPT

bArray - byte array containing algorithm specific initialization info.

bOff - offset within bArray where the algorithm specific data begins.

bLen - byte length of algorithm specific parameter data

Throws:

CryptoException - with the following reason codes:

- `CryptoException.ILLEGAL_VALUE` if theMode option is an undefined value or if a byte array parameter option is not supported by the algorithm or if the bLen is an incorrect byte length for the algorithm specific data or if the Key is inconsistent with the Cipher implementation.

getAlgorithm

```
public abstract byte getAlgorithm()
```

Gets the Cipher algorithm.

Returns:

the algorithm code defined above.

doFinal

```
public abstract short doFinal(byte[] inBuff,
                             short inOffset,
                             short inLength,
                             byte[] outBuff,
                             short outOffset)
    throws CryptoException
```

Generates encrypted/decrypted output from all/last input data. A call to this method also resets this Cipher object to the state it was in when previously initialized via a call to `init()`. That is, the object is reset and available to encrypt or decrypt (depending on the operation mode that was specified in the call to `init()`) more data.

The input and output buffer data may overlap.

Notes:

- On decryption operations (except when ISO 9797 method 1 padding is used), the padding bytes are not written to `outBuff`.
- On encryption operations, the number of bytes output into `outBuff` may be larger than `inLength`.

Parameters:

`inBuff` - the input buffer of data to be encrypted/decrypted.
`inOffset` - the offset into the input buffer at which to begin encryption/decryption.
`inLength` - the byte length to be encrypted/decrypted.
`outBuff` - the output buffer, may be the same as the input buffer
`outOffset` - the offset into the output buffer where the resulting hash value begins

Returns:

number of bytes output in `outBuff`

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
 - `CryptoException.INVALID_INIT` if this Cipher object is not initialized.
 - `CryptoException.ILLEGAL_USE` if this Cipher algorithm does not pad the message and the message is not block aligned or if the input message length is not supported.
-

update

```
public abstract short update(byte[] inBuff,
                             short inOffset,
```



```

        short inLength,
        byte[] outBuff,
        short outOffset)
    throws CryptoException

```

Generates encrypted/decrypted output from input data. When this method is used temporary storage of intermediate results is required. This method should only be used if all the input data required for the cipher is not available in one byte array. The `doFinal()` method is recommended whenever possible.

The input and output buffer data may overlap.

Notes:

- *On decryption operations(except when ISO 9797 method 1 padding is used), the padding bytes are not written to outBuff.*
- *On encryption operations, the number of bytes output into outBuff may be larger than inLength.*
- *On encryption and decryption operations(except when ISO 9797 method 1 padding is used), block alignment considerations may require that the number of bytes output into outBuff be smaller than inLength or even 0.*

Parameters:

`inBuff` - the input buffer of data to be encrypted/decrypted.

`inOffset` - the offset into the input buffer at which to begin encryption/decryption.

`inLength` - the byte length to be encrypted/decrypted.

`outBuff` - the output buffer, may be the same as the input buffer

`outOffset` - the offset into the output buffer where the resulting hash value begins

Returns:

number of bytes output in `outBuff`

Throws:

`CryptoException` - with the following reason codes:

- `CryptoException.UNINITIALIZED_KEY` if key not initialized.
 - `CryptoException.INVALID_INIT` if this `Cipher` object is not initialized.
 - `CryptoException.ILLEGAL_USE` if the input message length is not supported.
-

javacardx.crypto

Interface KeyEncryption

public abstract interface **KeyEncryption**

KeyEncryption interface defines the methods used to enable encrypted key data access to a key implementation.

See Also:

KeyBuilder, Cipher

Method Summary

Cipher	getKeyCipher() Returns the Cipher object to be used to decrypt the input key data and key parameters in the set methods. Default is null - no decryption performed.
void	setKeyCipher(Cipher keyCipher) Sets the Cipher object to be used to decrypt the input key data and key parameters in the set methods. Default Cipher object is null - no decryption performed.

Method Detail

setKeyCipher

public void **setKeyCipher**(Cipher keyCipher)

Sets the Cipher object to be used to decrypt the input key data and key parameters in the set methods.

Default Cipher object is null - no decryption performed.

Parameters:

keyCipher - the decryption Cipher object to decrypt the input key data. null parameter indicates that no decryption is required.

getKeyCipher

public Cipher **getKeyCipher()**

Returns the Cipher object to be used to decrypt the input key data and key parameters in the set methods.

Default is null - no decryption performed.

Returns:

keyCipher the decryption Cipher object to decrypt the input key data. null return indicates that no decryption is performed.

A B C D E G I J K L M N O P R S T U V W

A

abortTransaction() - Static method in class `javacard.framework.JCSystem`

Aborts the atomic transaction.

AID - class `javacard.framework.AID`.

This class encapsulates the Application Identifier(AID) associated with an applet.

AID(byte[], short, byte) - Constructor for class `javacard.framework.AID`

The JCRE uses this constructor to create a new AID instance encapsulating the specified AID bytes.

ALG_DES_CBC_ISO9797_M1 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_CBC_ISO9797_M1` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_CBC_ISO9797_M2 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_CBC_ISO9797_M2` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_CBC_NOPAD - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_CBC_NOPAD` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.

ALG_DES_CBC_PKCS5 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_CBC_PKCS5` provides a cipher using DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

ALG_DES_ECB_ISO9797_M1 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_ECB_ISO9797_M1` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_ECB_ISO9797_M2 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_ECB_ISO9797_M2` provides a cipher using DES in ECB mode. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_ECB_NOPAD - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_ECB_NOPAD` provides a cipher using DES in ECB mode. This algorithm does not pad input data.

ALG_DES_ECB_PKCS5 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_DES_ECB_PKCS5` provides a cipher using DES in ECB mode. Input data is padded according to the PKCS#5 scheme.

ALG_DES_MAC4_ISO9797_M1 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC4_ISO9797_M1` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_MAC4_ISO9797_M2 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC4_ISO9797_M2` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_MAC4_NOPAD - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC4_NOPAD` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.

ALG_DES_MAC4_PKCS5 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC4_PKCS5` generates a 4 byte MAC (most significant 4 bytes of encrypted block) using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

ALG_DES_MAC8_ISO9797_M1 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC8_ISO9797_M1` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 1 scheme.

ALG_DES_MAC8_ISO9797_M2 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC8_ISO9797_M2` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.

ALG_DES_MAC8_NOPAD - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC_8_NOPAD` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. This algorithm does not pad input data.

ALG_DES_MAC8_PKCS5 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DES_MAC8_PKCS5` generates a 8 byte MAC using DES or triple DES in CBC mode. This algorithm uses outer CBC for triple DES. Input data is padded according to the PKCS#5 scheme.

ALG_DSA_SHA - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_DSA_SHA` signs/verifies the 20 byte SHA digest using DSA.

ALG_MD5 - Static variable in class `javacard.security.MessageDigest`

Message Digest algorithm MD5.

ALG_PSEUDO_RANDOM - Static variable in class `javacard.security.RandomData`

Utility pseudo random number generation algorithms.

ALG_RIPEMD160 - Static variable in class `javacard.security.MessageDigest`

Message Digest algorithm RIPE MD-160.

ALG_RSA_ISO14888 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_RSA_ISO14888` provides a cipher using RSA. Input data is padded according to the ISO 14888 scheme.

ALG_RSA_ISO9796 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_RSA_ISO9796` provides a cipher using RSA. Input data is padded according to the ISO 9796 (EMV'96) scheme.

ALG_RSA_MD5_PKCS1 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_MD5_PKCS1` encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_RSA_MD5_RFC2409 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_MD5_RFC2409` encrypts the 16 byte MD5 digest using RSA. The digest is padded according to the RFC2409 scheme.

ALG_RSA_PKCS1 - Static variable in class `javacardx.crypto.Cipher`

Cipher algorithm `ALG_RSA_PKCS1` provides a cipher using RSA. Input data is padded according to

the PKCS#1 (v1.5) scheme.

ALG_RSA_RIPEMD160_ISO9796 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_RIPEMD160_ISO9796` encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the ISO 9796 scheme.

ALG_RSA_RIPEMD160_PKCS1 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_RIPEMD160_PKCS1` encrypts the 20 byte RIPE MD-160 digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_RSA_SHA_ISO9796 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_SHA_ISO9796` encrypts the 20 byte SHA digest using RSA. The digest is padded according to the ISO 9796 (EMV'96) scheme.

ALG_RSA_SHA_PKCS1 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_SHA_PKCS1` encrypts the 20 byte SHA digest using RSA. The digest is padded according to the PKCS#1 (v1.5) scheme.

ALG_RSA_SHA_RFC2409 - Static variable in class `javacard.security.Signature`

Signature algorithm `ALG_RSA_SHA_RFC2409` encrypts the 20 byte SHA digest using RSA. The digest is padded according to the RFC2409 scheme.

ALG_SECURE_RANDOM - Static variable in class `javacard.security.RandomData`

Cryptographically secure random number generation algorithms.

ALG_SHA - Static variable in class `javacard.security.MessageDigest`

Message Digest algorithm SHA.

APDU - class `javacard.framework.APDU`.

Application Protocol Data Unit (APDU) is the communication format between the card and the off-card applications.

APDUException - exception `javacard.framework.APDUException`.

`APDUException` represents an APDU related exception.

APDUException(short) - Constructor for class `javacard.framework.APDUException`

Constructs an `APDUException`.

Applet - class `javacard.framework.Applet`.

This abstract class defines an applet in Java Card.

Applet() - Constructor for class `javacard.framework.Applet`

Only this class's `install()` method should create the applet object.

ArithmeticException - exception `java.lang.ArithmeticException`.

A JCRE owned instance of `ArithmeticException` is thrown when an exceptional arithmetic condition has occurred.

ArithmeticException() - Constructor for class `java.lang.ArithmeticException`

Constructs an `ArithmeticException`.

arrayCompare(byte[], short, byte[], short, short) - Static method in class `javacard.framework.Util`

Compares an array from the specified source array, beginning at the specified position, with the specified position of the destination array from left to right.

arrayCopy(byte[], short, byte[], short, short) - Static method in class `javacard.framework.Util`

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

arrayCopyNonAtomic(byte[], short, byte[], short, short) - Static method in class `javacard.framework.Util`

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array (non-atomically).

arrayFillNonAtomic(byte[], short, short, byte) - Static method in class `javacard.framework.Util`
 Fills the byte array (non-atomically) beginning at the specified position, for the specified length with the specified byte value.

ArrayIndexOutOfBoundsException - exception `java.lang.ArrayIndexOutOfBoundsException`.
 A JCRE owned instance of `IndexOutOfBoundsException` is thrown to indicate that an array has been accessed with an illegal index.

ArrayIndexOutOfBoundsException() - Constructor for class `java.lang.ArrayIndexOutOfBoundsException`
 Constructs an `ArrayIndexOutOfBoundsException`.

ArrayStoreException - exception `java.lang.ArrayStoreException`.
 A JCRE owned instance of `ArrayStoreException` is thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

ArrayStoreException() - Constructor for class `java.lang.ArrayStoreException`
 Constructs an `ArrayStoreException`.

B

BAD_LENGTH - Static variable in class `javacard.framework.APDUException`
 This reason code is used by the `APDU.setOutgoingLength()` method to indicate that the length parameter is greater than 256 or if non BLOCK CHAINED data transfer is requested and `len` is greater than (IFSD-2), where IFSD is the Outgoing Block Size.

beginTransaction() - Static method in class `javacard.framework.JCSystem`
 Begins an atomic transaction.

BUFFER_BOUNDS - Static variable in class `javacard.framework.APDUException`
 This reason code is used by the `APDU.sendBytes()` method to indicate that the sum of buffer offset parameter and the byte length parameter exceeds the APDU buffer size.

BUFFER_FULL - Static variable in class `javacard.framework.TransactionException`
 This reason code is used during a transaction to indicate that the commit buffer is full.

buildKey(byte, short, boolean) - Static method in class `javacard.security.KeyBuilder`
 Creates cryptographic keys for signature and cipher algorithms.

C

CardException - exception `javacard.framework.CardException`.
 The `CardException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`.

CardException(short) - Constructor for class `javacard.framework.CardException`
 Construct a `CardException` instance with the specified reason.

CardRuntimeException - exception `javacard.framework.CardRuntimeException`.
 The `CardRuntimeException` class defines a field `reason` and two accessor methods `getReason()` and `setReason()`.

CardRuntimeException(short) - Constructor for class `javacard.framework.CardRuntimeException`
 Construct a `CardRuntimeException` instance with the specified reason.

check(byte[], short, byte) - Method in class javacard.framework.OwnerPIN
Compares pin against the PIN value.

check(byte[], short, byte) - Method in interface javacard.framework.PIN
Compares pin against the PIN value.

Cipher - class javacardx.crypto.Cipher.
The Cipher class is the abstract base class for Cipher algorithms.

Cipher() - Constructor for class javacardx.crypto.Cipher
Protected Constructor

CLA_ISO7816 - Static variable in interface javacard.framework.ISO7816
APDU command CLA : ISO 7816 = 0x00

ClassCastException - exception java.lang.ClassCastException.
A JCRE owned instance of ClassCastException is thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.

ClassCastException() - Constructor for class java.lang.ClassCastException
Constructs a ClassCastException.

CLEAR_ON_DESELECT - Static variable in class javacard.framework.JCSystem
This event code indicates that the contents of the transient object are cleared to the default value on applet deselection event or in CLEAR_ON_RESET cases.

CLEAR_ON_RESET - Static variable in class javacard.framework.JCSystem
This event code indicates that the contents of the transient object are cleared to the default value on card reset (or power on) event.

clearKey() - Method in interface javacard.security.Key
Clears the key and sets its initialized state to false.

commitTransaction() - Static method in class javacard.framework.JCSystem
Commits an atomic transaction.

CryptoException - exception javacard.security.CryptoException.
CryptoException represents a cryptography-related exception.

CryptoException(short) - Constructor for class javacard.security.CryptoException
Constructs a CryptoException with the specified reason.

D

deselect() - Method in class javacard.framework.Applet
Called by the JCRE to inform this currently selected applet that another (or the same) applet will be selected.

DESKey - interface javacard.security.DESKey.
DESKey contains an 8/16/24 byte key for single/2 key triple DES/3 key triple DES operations.

doFinal(byte[], short, short, byte[], short) - Method in class javacard.security.MessageDigest
Generates a hash of all/last input data.

doFinal(byte[], short, short, byte[], short) - Method in class javacardx.crypto.Cipher
Generates encrypted/decrypted output from all/last input data.

DSAKey - interface javacard.security.DSAKey.
The DSAKey interface is the base interface for the DSA algorithms private and public key implementations.

DSAPrivateKey - interface javacard.security.DSAPrivateKey.

The DSAPrivateKey interface is used to sign data using the DSA algorithm.

DSAPublicKey - interface javacard.security.DSAPublicKey.

The DSAPublicKey interface is used to verify signatures on signed data using the DSA algorithm.

E

equals(byte[], short, byte) - Method in class javacard.framework.AID

Checks if the specified AID bytes in bArray are the same as those encapsulated in this AID object.

equals(Object) - Method in class java.lang.Object

Compares two Objects for equality.

equals(Object) - Method in class javacard.framework.AID

Compares the AID bytes in this AID instance to the AID bytes in the specified object.

Exception - exception java.lang.Exception.

The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable applet might want to catch.

Exception() - Constructor for class java.lang.Exception

Constructs an Exception instance.

G

generateData(byte[], short, short) - Method in class javacard.security.RandomData

Generates random data.

getAID() - Static method in class javacard.framework.JCSystem

Returns the JCRE owned instance of the AID object associated with the current applet context.

getAlgorithm() - Method in class javacard.security.MessageDigest

Gets the Message digest algorithm.

getAlgorithm() - Method in class javacard.security.Signature

Gets the Signature algorithm.

getAlgorithm() - Method in class javacardx.crypto.Cipher

Gets the Cipher algorithm.

getAppletShareableInterfaceObject(AID, byte) - Static method in class javacard.framework.JCSystem

This method is called by a client applet to get a server applet's shareable interface object.

getBuffer() - Method in class javacard.framework.APDU

Returns the APDU buffer byte array.

getBytes(byte[], short) - Method in class javacard.framework.AID

Called to get the AID bytes encapsulated within AID object.

getDP1(byte[], short) - Method in interface javacard.security.RSAPrivateCrtKey

Returns the value of the DP1 parameter in plain text.

getDQ1(byte[], short) - Method in interface javacard.security.RSAPrivateCrtKey

Returns the value of the DQ1 parameter in plain text.

- getExponent(byte[], short)** - Method in interface `javacard.security.RSAPrivateKey`
Returns the private exponent value of the key in plain text.
- getExponent(byte[], short)** - Method in interface `javacard.security.RSAPublicKey`
Returns the private exponent value of the key in plain text.
- getG(byte[], short)** - Method in interface `javacard.security.DSAKey`
Returns the subprime parameter value of the key in plain text.
- getInBlockSize()** - Static method in class `javacard.framework.APDU`
Returns the configured incoming block size. In T=1 protocol, this corresponds to IFSC (information field size for ICC), the maximum size of incoming data blocks into the card. In T=0 protocol, this method returns 1.
- getInstance(byte)** - Static method in class `javacard.security.RandomData`
Creates a `RandomData` instance of the selected algorithm.
- getInstance(byte, boolean)** - Static method in class `javacard.security.MessageDigest`
Creates a `MessageDigest` object instance of the selected algorithm.
- getInstance(byte, boolean)** - Static method in class `javacard.security.Signature`
Creates a `Signature` object instance of the selected algorithm.
- getInstance(byte, boolean)** - Static method in class `javacardx.crypto.Cipher`
Creates a `Cipher` object instance of the selected algorithm.
- getKey(byte[], short)** - Method in interface `javacard.security.DESKey`
Returns the Key data in plain text.
- getKeyCipher()** - Method in interface `javacardx.crypto.KeyEncryption`
Returns the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods. Default is null - no decryption performed.
- getLength()** - Method in class `javacard.security.MessageDigest`
Returns the byte length of the hash.
- getLength()** - Method in class `javacard.security.Signature`
Returns the byte length of the signature data.
- getMaxCommitCapacity()** - Static method in class `javacard.framework.JCSystem`
Returns the total number of bytes in the commit buffer.
- getModulus(byte[], short)** - Method in interface `javacard.security.RSAPrivateKey`
Returns the modulus value of the key in plain text.
- getModulus(byte[], short)** - Method in interface `javacard.security.RSAPublicKey`
Returns the modulus value of the key in plain text.
- getNAD()** - Method in class `javacard.framework.APDU`
In T=1 protocol, this method returns the Node Address byte, NAD. In T=0 protocol, this method returns 0.
- getOutBlockSize()** - Static method in class `javacard.framework.APDU`
Returns the configured outgoing block size. In T=1 protocol, this corresponds to IFSD (information field size for interface device), the maximum size of outgoing data blocks to the CAD. In T=0 protocol, this method returns 258 (accounts for 2 status bytes).
- getP(byte[], short)** - Method in interface `javacard.security.DSAKey`
Returns the base parameter value of the key in plain text.
- getP(byte[], short)** - Method in interface `javacard.security.RSAPrivateCrtKey`
Returns the value of the P parameter in plain text.
- getPQ(byte[], short)** - Method in interface `javacard.security.RSAPrivateCrtKey`
Returns the value of the PQ parameter in plain text.

- getPreviousContextAID()** - Static method in class `javacard.framework.JCSystem`
This method is called to obtain the JCRE owned instance of the AID object associated with the previously active applet context.
- getProtocol()** - Static method in class `javacard.framework.APDU`
Returns the ISO 7816 transport protocol type, T=1 or T=0 in progress.
- getQ(byte[], short)** - Method in interface `javacard.security.DSAKey`
Returns the prime parameter value of the key in plain text.
- getQ(byte[], short)** - Method in interface `javacard.security.RSAPrivateCrtKey`
Returns the value of the Q parameter in plain text.
- getReason()** - Method in class `javacard.framework.CardRuntimeException`
Get reason code
- getReason()** - Method in class `javacard.framework.CardException`
Get reason code
- getShareableInterfaceObject(AID, byte)** - Method in class `javacard.framework.Applet`
Called by the JCRE to obtain a shareable interface object from this server applet, on behalf of a request from a client applet.
- getShort(byte[], short)** - Static method in class `javacard.framework.Util`
Concatenates two bytes in a byte array to form a short value.
- getSize()** - Method in interface `javacard.security.Key`
Returns the key size in number of bits.
- getTransactionDepth()** - Static method in class `javacard.framework.JCSystem`
Returns the current transaction nesting depth level.
- getTriesRemaining()** - Method in class `javacard.framework.OwnerPIN`
Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.
- getTriesRemaining()** - Method in interface `javacard.framework.PIN`
Returns the number of times remaining that an incorrect PIN can be presented before the PIN is blocked.
- getType()** - Method in interface `javacard.security.Key`
Returns the key interface type.
- getUnusedCommitCapacity()** - Static method in class `javacard.framework.JCSystem`
Returns the number of bytes left in the commit buffer.
- getValidatedFlag()** - Method in class `javacard.framework.OwnerPIN`
This protected method returns the validated flag.
- getVersion()** - Static method in class `javacard.framework.JCSystem`
Returns the current major and minor version of the Java Card API.
- getX(byte[], short)** - Method in interface `javacard.security.DSAPrivateKey`
Returns the value of the key in plain text.
- getY(byte[], short)** - Method in interface `javacard.security.DSAPublicKey`
Returns the value of the key in plain text.
-

I

ILLEGAL_AID - Static variable in class `javacard.framework.SystemException`

This reason code is used by the `javacard.framework.Applet.register()` method to indicate that the input AID parameter is not a legal AID value.

ILLEGAL_TRANSIENT - Static variable in class `javacard.framework.SystemException`

This reason code is used to indicate that the request to create a transient object is not allowed in the current applet context.

ILLEGAL_USE - Static variable in class `javacard.framework.APDUEException`

This `APDUEException` reason code indicates that the method should not be invoked based on the current state of the APDU.

ILLEGAL_USE - Static variable in class `javacard.security.CryptoException`

This reason code is used to indicate that the signature or cipher algorithm does not pad the incoming message and the input message is not block aligned.

ILLEGAL_VALUE - Static variable in class `javacard.framework.PINException`

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

ILLEGAL_VALUE - Static variable in class `javacard.framework.SystemException`

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

ILLEGAL_VALUE - Static variable in class `javacard.security.CryptoException`

This reason code is used to indicate that one or more input parameters is out of allowed bounds.

IN_PROGRESS - Static variable in class `javacard.framework.TransactionException`

This reason code is used by the `beginTransaction` method to indicate a transaction is already in progress.

IndexOutOfBoundsException - exception `java.lang.IndexOutOfBoundsException`.

A JCRE owned instance of `IndexOutOfBoundsException` is thrown to indicate that an index of some sort (such as to an array) is out of range.

IndexOutOfBoundsException() - Constructor for class `java.lang.IndexOutOfBoundsException`

Constructs an `IndexOutOfBoundsException`.

init(Key, byte) - Method in class `javacard.security.Signature`

Initializes the `Signature` object with the appropriate Key.

init(Key, byte) - Method in class `javacardx.crypto.Cipher`

Initializes the `Cipher` object with the appropriate Key.

init(Key, byte, byte[], short, short) - Method in class `javacard.security.Signature`

Initializes the `Signature` object with the appropriate Key and algorithm specific parameters.

init(Key, byte, byte[], short, short) - Method in class `javacardx.crypto.Cipher`

Initializes the `Cipher` object with the appropriate Key and algorithm specific parameters.

INS_EXTERNAL_AUTHENTICATE - Static variable in interface `javacard.framework.ISO7816`

APDU command `INS : EXTERNAL AUTHENTICATE = 0x82`

INS_SELECT - Static variable in interface `javacard.framework.ISO7816`

APDU command `INS : SELECT = 0xA4`

install(byte[], short, byte) - Static method in class `javacard.framework.Applet`

To create an instance of the `Applet` subclass, the JCRE will call this static method first.

INTERNAL_FAILURE - Static variable in class `javacard.framework.TransactionException`

This reason code is used during a transaction to indicate an internal JCRE problem (fatal error).

INVALID_INIT - Static variable in class `javacard.security.CryptoException`

This reason code is used to indicate that the signature or cipher object has not been correctly initialized for the requested operation.

IO_ERROR - Static variable in class `javacard.framework.APDUException`

This reason code indicates that an unrecoverable error occurred in the I/O transmission layer.

isInitialized() - Method in interface `javacard.security.Key`

Reports the initialized state of the key.

ISO7816 - interface `javacard.framework.ISO7816`.

ISO7816 encapsulates constants related to ISO 7816-3 and ISO 7816-4.

ISOException - exception `javacard.framework.ISOException`.

`ISOException` class encapsulates an ISO 7816-4 response status word as its reason code.

ISOException(short) - Constructor for class `javacard.framework.ISOException`

Constructs an `ISOException` instance with the specified status word.

isTransient(Object) - Static method in class `javacard.framework.JCSystem`

Used to check if the specified object is transient.

isValidated() - Method in class `javacard.framework.OwnerPIN`

Returns `true` if a valid PIN has been presented since the last card reset or last call to `reset()`.

isValidated() - Method in interface `javacard.framework.PIN`

Returns `true` if a valid PIN value has been presented since the last card reset or last call to `reset()`.

J

`java.lang` - package `java.lang`

Provides classes that are fundamental to the design of the Java Card technology subset of the Java programming language.

`javacard.framework` - package `javacard.framework`

Provides framework of classes and interfaces for the core functionality of a Java Card applet.

`javacard.security` - package `javacard.security`

Provides the classes and interfaces for the Java Card security framework.

`javacardx.crypto` - package `javacardx.crypto`

Extension package containing security classes and interfaces for export-controlled functionality.

JCSystem - class `javacard.framework.JCSystem`.

The `JCSystem` class includes a collection of methods to control applet execution, resource management, atomic transaction management and inter-applet object sharing in Java Card.

K

Key - interface `javacard.security.Key`.

The `Key` interface is the base interface for all keys.

KeyBuilder - class `javacard.security.KeyBuilder`.

The `KeyBuilder` class is a key object factory.

KeyEncryption - interface `javacardx.crypto.KeyEncryption`.

`KeyEncryption` interface defines the methods used to enable encrypted key data access to a key implementation.

L

LENGTH_DES - Static variable in class `javacard.security.KeyBuilder`

DES Key Length `LENGTH_DES` = 64.

LENGTH_DES3_2KEY - Static variable in class `javacard.security.KeyBuilder`

DES Key Length `LENGTH_DES3_2KEY` = 128.

LENGTH_DES3_3KEY - Static variable in class `javacard.security.KeyBuilder`

DES Key Length `LENGTH_DES3_3KEY` = 192.

LENGTH_DSA_1024 - Static variable in class `javacard.security.KeyBuilder`

DSA Key Length `LENGTH_DSA_1024` = 1024.

LENGTH_DSA_512 - Static variable in class `javacard.security.KeyBuilder`

DSA Key Length `LENGTH_DSA_512` = 512.

LENGTH_DSA_768 - Static variable in class `javacard.security.KeyBuilder`

DSA Key Length `LENGTH_DSA_768` = 768.

LENGTH_RSA_1024 - Static variable in class `javacard.security.KeyBuilder`

RSA Key Length `LENGTH_RSA_1024` = 1024.

LENGTH_RSA_2048 - Static variable in class `javacard.security.KeyBuilder`

RSA Key Length `LENGTH_RSA_2048` = 2048.

LENGTH_RSA_512 - Static variable in class `javacard.security.KeyBuilder`

RSA Key Length `LENGTH_RSA_512` = 512.

LENGTH_RSA_768 - Static variable in class `javacard.security.KeyBuilder`

RSA Key Length `LENGTH_RSA_768` = 768.

lookupAID(byte[], short, byte) - Static method in class `javacard.framework.JCSystem`

Returns the JCRE owned instance of the AID object, if any, encapsulating the specified AID bytes in the `buffer` parameter if there exists a successfully installed applet on the card whose instance AID exactly matches that of the specified AID bytes.

M

makeShort(byte, byte) - Static method in class `javacard.framework.Util`

Concatenates the two parameter bytes to form a short value.

makeTransientBooleanArray(short, byte) - Static method in class `javacard.framework.JCSystem`

Create a transient boolean array with the specified array length.

makeTransientByteArray(short, byte) - Static method in class `javacard.framework.JCSystem`

Create a transient byte array with the specified array length.

makeTransientObjectArray(short, byte) - Static method in class `javacard.framework.JCSystem`

Create a transient array of `Object` with the specified array length.

makeTransientShortArray(short, byte) - Static method in class `javacard.framework.JCSystem`

Create a transient short array with the specified array length.

MessageDigest - class `javacard.security.MessageDigest`.

The `MessageDigest` class is the base class for hashing algorithms.

MessageDigest() - Constructor for class `javacard.security.MessageDigest`

Protected Constructor

MODE_DECRYPT - Static variable in class `javacardx.crypto.Cipher`

Used in `init()` methods to indicate decryption mode.

MODE_ENCRYPT - Static variable in class `javacardx.crypto.Cipher`

Used in `init()` methods to indicate encryption mode.

MODE_SIGN - Static variable in class `javacard.security.Signature`

Used in `init()` methods to indicate signature sign mode.

MODE_VERIFY - Static variable in class `javacard.security.Signature`

Used in `init()` methods to indicate signature verify mode.

N

NegativeArraySizeException - exception `java.lang.NegativeArraySizeException`.

A JCRE owned instance of `NegativeArraySizeException` is thrown if an applet tries to create an array with negative size.

NegativeArraySizeException() - Constructor for class `java.lang.NegativeArraySizeException`

Constructs a `NegativeArraySizeException`.

NO_RESOURCE - Static variable in class `javacard.framework.SystemException`

This reason code is used to indicate that there is insufficient resource in the Card for the request.

NO_SUCH_ALGORITHM - Static variable in class `javacard.security.CryptoException`

This reason code is used to indicate that the requested algorithm or key type is not supported.

NO_T0_GETRESPONSE - Static variable in class `javacard.framework.APDUException`

This reason code indicates that during T=0 protocol, the CAD did not return a GET RESPONSE command in response to a <61xx> response status to send additional data.

NO_TRANSIENT_SPACE - Static variable in class `javacard.framework.SystemException`

This reason code is used by the `makeTransient...()` methods to indicate that no room is available in volatile memory for the requested object.

NOT_A_TRANSIENT_OBJECT - Static variable in class `javacard.framework.JCSystem`

This event code indicates that the object is not transient.

NOT_IN_PROGRESS - Static variable in class `javacard.framework.TransactionException`

This reason code is used by the `abortTransaction` and `commitTransaction` methods when a transaction is not in progress.

NullPointerException - exception `java.lang.NullPointerException`.

A JCRE owned instance of `NullPointerException` is thrown when an applet attempts to use null in a case where an object is required.

NullPointerException() - Constructor for class `java.lang.NullPointerException`

Constructs a `NullPointerException`.

O

Object - class java.lang.Object.

Class `Object` is the root of the Java Card class hierarchy.

Object() - Constructor for class java.lang.Object

OFFSET_CDATA - Static variable in interface javacard.framework.ISO7816

APDU command data offset : CDATA = 5

OFFSET_CLA - Static variable in interface javacard.framework.ISO7816

APDU header offset : CLA = 0

OFFSET_INS - Static variable in interface javacard.framework.ISO7816

APDU header offset : INS = 1

OFFSET_LC - Static variable in interface javacard.framework.ISO7816

APDU header offset : LC = 4

OFFSET_P1 - Static variable in interface javacard.framework.ISO7816

APDU header offset : P1 = 2

OFFSET_P2 - Static variable in interface javacard.framework.ISO7816

APDU header offset : P2 = 3

OwnerPIN - class javacard.framework.OwnerPIN.

This class represents an Owner PIN.

OwnerPIN(byte, byte) - Constructor for class javacard.framework.OwnerPIN

Constructor.

P

partialEquals(byte[], short, byte) - Method in class javacard.framework.AID

Checks if the specified partial AID byte sequence matches the first `length` bytes of the encapsulated AID bytes within this AID object.

PIN - interface javacard.framework.PIN.

This interface represents a PIN.

PINException - exception javacard.framework.PINException.

`PINException` represents a `OwnerPIN` class access-related exception.

PINException(short) - Constructor for class javacard.framework.PINException

Constructs a `PINException`.

PrivateKey - interface javacard.security.PrivateKey.

The `PrivateKey` class is the base class for private keys used in asymmetric algorithms.

process(APDU) - Method in class javacard.framework.Applet

Called by the JCRE to process an incoming APDU command.

PROTOCOL_T0 - Static variable in class javacard.framework.APDU

ISO 7816 transport protocol type T=0

PROTOCOL_T1 - Static variable in class javacard.framework.APDU

ISO 7816 transport protocol type T=1

PublicKey - interface javacard.security.PublicKey.

The `PublicKey` class is the base class for public keys used in asymmetric algorithms.

R

RandomData - class javacard.security.RandomData.

The RandomData abstract class is the base class for random number generation.

RandomData() - Constructor for class javacard.security.RandomData

Protected constructor for subclassing.

receiveBytes(short) - Method in class javacard.framework.APDU

Gets as many data bytes as will fit without APDU buffer overflow, at the specified offset bOff.

Gets all the remaining bytes if they fit.

register() - Method in class javacard.framework.Applet

This method is used by the applet to register this applet instance with the JCRE and to assign the Applet subclass AID bytes as its instance AID bytes.

register(byte[], short, byte) - Method in class javacard.framework.Applet

This method is used by the applet to register this applet instance with the JCRE and assign the specified AID bytes as its instance AID bytes.

reset() - Method in class javacard.framework.OwnerPIN

If the validated flag is set, this method resets it.

reset() - Method in interface javacard.framework.PIN

If the validated flag is set, this method resets it.

reset() - Method in class javacard.security.MessageDigest

Resets the MessageDigest object to the initial state for further use.

resetAndUnblock() - Method in class javacard.framework.OwnerPIN

This method resets the validated flag and resets the PIN try counter to the value of the PIN try limit.

RIDEquals(AID) - Method in class javacard.framework.AID

Checks if the RID (National Registered Application provider identifier) portion of the encapsulated AID bytes within the otherAID object matches that of this AID object.

RSAPrivateCrtKey - interface javacard.security.RSAPrivateCrtKey.

The RSAPrivateCrtKey interface is used to sign data using the RSA algorithm in its Chinese Remainder Theorem form.

RSAPrivateKey - interface javacard.security.RSAPrivateKey.

The RSAPrivateKey class is used to sign data using the RSA algorithm in its modulus/exponent form.

RSAPublicKey - interface javacard.security.RSAPublicKey.

The RSAPublicKey is used to verify signatures on signed data using the RSA algorithm.

RuntimeException - exception java.lang.RuntimeException.

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Card Virtual Machine. A method is not required to declare in its throws clause any subclasses of RuntimeException that might be thrown during the execution of the method but not caught.

RuntimeException() - Constructor for class java.lang.RuntimeException

Constructs a RuntimeException instance.

S

SecretKey - interface `javacard.security.SecretKey`.

The `SecretKey` class is the base interface for keys used in symmetric algorithms (e.g. DES).

SecurityException - exception `java.lang.SecurityException`.

A JCRE owned instance of `SecurityException` is thrown by the Java Card Virtual Machine to indicate a security violation. This exception is thrown when an attempt is made to illegally access an object belonging to another applet.

SecurityException() - Constructor for class `java.lang.SecurityException`
Constructs a `SecurityException`.

select() - Method in class `javacard.framework.Applet`
Called by the JCRE to inform this applet that it has been selected.

selectingApplet() - Method in class `javacard.framework.Applet`
This method is used by the `process()` method to distinguish the SELECT APDU command which selected this applet, from all other other SELECT APDU commands which may relate to file or internal applet state selection.

sendBytes(short, short) - Method in class `javacard.framework.APDU`
Sends `len` more bytes from APDU buffer at specified offset `boff`.

sendBytesLong(byte[], short, short) - Method in class `javacard.framework.APDU`
Sends `len` more bytes from `outData` byte array starting at specified offset `boff`.

setDP1(byte[], short, short) - Method in interface `javacard.security.RSAPrivateCrtKey`
Sets the value of the DP1 parameter.

setDQ1(byte[], short, short) - Method in interface `javacard.security.RSAPrivateCrtKey`
Sets the value of the DQ1 parameter.

setExponent(byte[], short, short) - Method in interface `javacard.security.RSAPrivateKey`
Sets the private exponent value of the key.

setExponent(byte[], short, short) - Method in interface `javacard.security.RSAPublicKey`
Sets the public exponent value of the key.

setG(byte[], short, short) - Method in interface `javacard.security.DSAKey`
Sets the subprime parameter value of the key.

setIncomingAndReceive() - Method in class `javacard.framework.APDU`
This is the primary receive method.

setKey(byte[], short) - Method in interface `javacard.security.DESKey`
Sets the Key data.

setKeyCipher(Cipher) - Method in interface `javacardx.crypto.KeyEncryption`
Sets the `Cipher` object to be used to decrypt the input key data and key parameters in the set methods. Default `Cipher` object is `null` - no decryption performed.

setModulus(byte[], short, short) - Method in interface `javacard.security.RSAPrivateKey`
Sets the modulus value of the key.

setModulus(byte[], short, short) - Method in interface `javacard.security.RSAPublicKey`
Sets the modulus value of the key.

setOutgoing() - Method in class `javacard.framework.APDU`
This method is used to set the data transfer direction to outbound and to obtain the expected length of response (`Le`).

setOutgoingAndSend(short, short) - Method in class javacard.framework.APDU

This is the "convenience" send method.

setOutgoingLength(short) - Method in class javacard.framework.APDU

Sets the actual length of response data.

setOutgoingNoChaining() - Method in class javacard.framework.APDU

This method is used to set the data transfer direction to outbound without using BLOCK CHAINING(See ISO 7816-3/4) and to obtain the expected length of response (Le).

setP(byte[], short, short) - Method in interface javacard.security.DSAKey

Sets the base parameter value of the key.

setP(byte[], short, short) - Method in interface javacard.security.RSAPrivateCrtKey

Sets the value of the P parameter.

setPQ(byte[], short, short) - Method in interface javacard.security.RSAPrivateCrtKey

Sets the value of the PQ parameter.

setQ(byte[], short, short) - Method in interface javacard.security.DSAKey

Sets the prime parameter value of the key.

setQ(byte[], short, short) - Method in interface javacard.security.RSAPrivateCrtKey

Sets the value of the Q parameter.

setReason(short) - Method in class javacard.framework.CardRuntimeException

Set reason code

setReason(short) - Method in class javacard.framework.CardException

Set reason code

setSeed(byte[], short, short) - Method in class javacard.security.RandomData

Seeds the random data generator.

setShort(byte[], short, short) - Static method in class javacard.framework.Util

Deposits the short value as two successive bytes at the specified offset in the byte array.

setValidatedFlag(boolean) - Method in class javacard.framework.OwnerPIN

This protected method sets the value of the validated flag.

setX(byte[], short, short) - Method in interface javacard.security.DSAPrivateKey

Sets the value of the key.

setY(byte[], short, short) - Method in interface javacard.security.DSAPublicKey

Sets the value of the key.

Shareable - interface javacard.framework.Shareable.

The Shareable interface serves to identify all shared objects.

sign(byte[], short, short, byte[], short) - Method in class javacard.security.Signature

Generates the signature of all/last input data.

Signature - class javacard.security.Signature.

The Signature class is the base class for Signature algorithms.

Signature() - Constructor for class javacard.security.Signature

Protected Constructor

SW_APPLET_SELECT_FAILED - Static variable in interface javacard.framework.ISO7816

Response status : Applet selection failed = 0x6999;

SW_BYTES_REMAINING_00 - Static variable in interface javacard.framework.ISO7816

Response status : Response bytes remaining = 0x6100

SW_CLA_NOT_SUPPORTED - Static variable in interface javacard.framework.ISO7816

Response status : CLA value not supported = 0x6E00

SW_COMMAND_NOT_ALLOWED - Static variable in interface javacard.framework.ISO7816
Response status : Command not allowed (no current EF) = 0x6986

SW_CONDITIONS_NOT_SATISFIED - Static variable in interface javacard.framework.ISO7816
Response status : Conditions of use not satisfied = 0x6985

SW_CORRECT_LENGTH_00 - Static variable in interface javacard.framework.ISO7816
Response status : Correct Expected Length (Le) = 0x6C00

SW_DATA_INVALID - Static variable in interface javacard.framework.ISO7816
Response status : Data invalid = 0x6984

SW_FILE_FULL - Static variable in interface javacard.framework.ISO7816
Response status : Not enough memory space in the file = 0x6A84

SW_FILE_INVALID - Static variable in interface javacard.framework.ISO7816
Response status : File invalid = 0x6983

SW_FILE_NOT_FOUND - Static variable in interface javacard.framework.ISO7816
Response status : File not found = 0x6A82

SW_FUNC_NOT_SUPPORTED - Static variable in interface javacard.framework.ISO7816
Response status : Function not supported = 0x6A81

SW_INCORRECT_P1P2 - Static variable in interface javacard.framework.ISO7816
Response status : Incorrect parameters (P1,P2) = 0x6A86

SW_INS_NOT_SUPPORTED - Static variable in interface javacard.framework.ISO7816
Response status : INS value not supported = 0x6D00

SW_NO_ERROR - Static variable in interface javacard.framework.ISO7816
Response status : No Error = (short)0x9000

SW_RECORD_NOT_FOUND - Static variable in interface javacard.framework.ISO7816
Response status : Record not found = 0x6A83

SW_SECURITY_STATUS_NOT_SATISFIED - Static variable in interface javacard.framework.ISO7816
Response status : Security condition not satisfied = 0x6982

SW_UNKNOWN - Static variable in interface javacard.framework.ISO7816
Response status : No precise diagnosis = 0x6F00

SW_WRONG_DATA - Static variable in interface javacard.framework.ISO7816
Response status : Wrong data = 0x6A80

SW_WRONG_LENGTH - Static variable in interface javacard.framework.ISO7816
Response status : Wrong length = 0x6700

SW_WRONG_P1P2 - Static variable in interface javacard.framework.ISO7816
Response status : Incorrect parameters (P1,P2) = 0x6B00

SystemException - exception javacard.framework.SystemException.
SystemException represents a JCSysm class related exception.

SystemException(short) - Constructor for class javacard.framework.SystemException
Constructs a SystemException.

T

T1_IFD_ABORT - Static variable in class javacard.framework.APDUEException
This reason code indicates that during T=1 protocol, the CAD returned an ABORT S-Block command and aborted the data transfer.

Throwable - class java.lang.Throwable.

The Throwable class is the superclass of all errors and exceptions in the Java Card subset of the Java language.

Throwable() - Constructor for class java.lang.Throwable

Constructs a new Throwable.

throwIt(short) - Static method in class javacard.framework.CardRuntimeException

Throw the JCRE owned instance of the CardRuntimeException class with the specified reason.

throwIt(short) - Static method in class javacard.framework.PINException

Throws the JCRE owned instance of PINException with the specified reason.

throwIt(short) - Static method in class javacard.framework.ISOException

Throws the JCRE owned instance of the ISOException class with the specified status word.

throwIt(short) - Static method in class javacard.framework.CardException

Throw the JCRE owned instance of CardException class with the specified reason.

throwIt(short) - Static method in class javacard.framework.UserException

Throws the JCRE owned instance of UserException with the specified reason.

throwIt(short) - Static method in class javacard.framework.SystemException

Throws the JCRE owned instance of SystemException with the specified reason.

throwIt(short) - Static method in class javacard.framework.TransactionException

Throws the JCRE owned instance of TransactionException with the specified reason.

throwIt(short) - Static method in class javacard.framework.APDUException

Throws the JCRE owned instance of APDUException with the specified reason.

throwIt(short) - Static method in class javacard.security.CryptoException

Throws the JCRE owned instance of CryptoException with the specified reason.

TransactionException - exception javacard.framework.TransactionException.

TransactionException represents an exception in the transaction subsystem.

TransactionException(short) - Constructor for class javacard.framework.TransactionException

Constructs a TransactionException with the specified reason.

TYPE_DES - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type DESKey with persistent key data.

TYPE_DES_TRANSIENT_DESELECT - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type DESKey with CLEAR_ON_DESELECT transient key data.

TYPE_DES_TRANSIENT_RESET - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type DESKey with CLEAR_ON_RESET transient key data.

TYPE_DSA_PRIVATE - Static variable in class javacard.security.KeyBuilder

Key object which implements the interface type DSAPrivateKey for the DSA algorithm.

TYPE_DSA_PUBLIC - Static variable in class javacard.security.KeyBuilder

Key object which implements the interface type DSAPublicKey for the DSA algorithm.

TYPE_RSA_CRT_PRIVATE - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type RSAPrivateCrtKey which uses Chinese Remainder Theorem.

TYPE_RSA_PRIVATE - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type RSAPrivateKey which uses modulus/exponent form.

TYPE_RSA_PUBLIC - Static variable in class javacard.security.KeyBuilder

Key object which implements interface type RSAPublicKey.

U

UNINITIALIZED_KEY - Static variable in class javacard.security.CryptoException

This reason code is used to indicate that the key is uninitialized.

update(byte[], short, byte) - Method in class javacard.framework.OwnerPIN

This method sets a new value for the PIN and resets the PIN try counter to the value of the PIN try limit.

update(byte[], short, short) - Method in class javacard.security.MessageDigest

Accumulates a hash of the input data.

update(byte[], short, short) - Method in class javacard.security.Signature

Accumulates a signature of the input data.

update(byte[], short, short, byte[], short) - Method in class javacardx.crypto.Cipher

Generates encrypted/decrypted output from input data.

UserException - exception javacard.framework.UserException.

UserException represents a User exception.

UserException() - Constructor for class javacard.framework.UserException

Constructs a UserException with reason = 0.

UserException(short) - Constructor for class javacard.framework.UserException

Constructs a UserException with the specified reason.

Util - class javacard.framework.Util.

The Util class contains common utility functions.

V

verify(byte[], short, short, byte[], short, short) - Method in class javacard.security.Signature

Verifies the signature of all/last input data against the passed in signature.

W

waitExtension() - Static method in class javacard.framework.APDU

Requests additional processing time from CAD.

A B C D E G I J K L M N O P R S T U V W
